WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Monday, July 05, 2004

Hide?	Set Name	Query	Hit Count
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
☐	L44	l43 and (load-balancing or (load adj3 balancing) or (load adj3 balance))	0
☐	L43	19991222	28
☐	L42	(replica or mirroring) adj3 agent	55
☐	L41	(DNS or (domain adj5 server)) same ((preference or priority) adj4 (indicator or index))	2
☐	L40	(DNS or (domain adj5 server)) same ((preference or priority) adj4 (indicator of index))	0
☐	L39	L38 and (load-balancing or (load adj3 balancing) or (load adj3 balance))	14
☐	L38	19991222	815
☐	L37	(mirror or mirrored or mirroring or replicate or replica)adj2 (table or list or record)	1471
☐	L36	L32 and l24	0
☐	L35	L32 and l20	0
☐	L34	L32 and l15	0
☐	L33	L32 and l1	0
☐	L32	19991222	105
☐	L31	(mirroring or replicate)adj2 table	203
☐	L30	(mirror or mirrored or mirroring or replicate)adj2 table	840
☐	L29	19991222	2335
☐	L28	(mirror or mirrored or mirroring or replicate) near4 table	4001
☐	L27	L25 and (mirror or mirrored or mirroring or replicate)	31
☐	L26	L25 and ((mirror or mirrored or mirroring) near5 (list or table))	0
☐	L25	19991222	230
☐	L24	distributed near5 (mail or e-mail or (electronic adj3 mail)) near5 system	421
☐	L23	(mail or e-mail or (electronic adj3 mail)) near8 server near8 ((mirror or mirrored or mirroring) near5 (list or table))	2
☐	L22	19991222	54
☐	L21	L20 and (load-balancing or (load adj3 balancing) or (load adj3 balance))	126
☐	L20	(mail or e-mail or (electronic adj3 mail)) near5 management	3652
☐	L19	L15 and ((mirror or mirrored or mirroring) near5 (list or table))	6
☐	L18	L17 and (mail adj3 exchange)	0

☐	L17	L15 and (mirror or mirrored or mirroring)	57
☐	L16	L15 and (mirror or mirrored)	47
☐	L15	L14 and (load-balancing or (load adj3 balancing) or (load adj3 balance))	241
☐	L14	19991222	983
☐	L13	(cluster or clustered or farm) near8 server	4147
☐	L12	L11 or l7 or l6	5
☐	L11	19991222	3
☐	L10	l1 and ((message or e-mail or mail) adj3 agent)	6
☐	L9	l1 and (message adj3 agent)	3
☐	L8	19991222	0
☐	L7	19991222	2
☐	L6	19991222	3
☐	L5	L1 and (mail adj3 exchange)	7
☐	L4	L1 and (mail adj3 exchande)	0
☐	L3	L1 and (mirror or mirrored)	10
☐	L2	L1 and (load-balancing or (load adj3 balancing) or (load adj3 balance))	17
☐	L1	(cluster or clustered or farm) near8 (mail or e-mail or (electronic adj3 mail))	148

END OF SEARCH HISTORY

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L12: Entry 1 of 5 File: USPT Jan 1, 2002

DOCUMENT-IDENTIFIER: US 6336138 B1
TITLE: Template-driven approach for generating models on network services

Application Filing Date (1):
19980825

Detailed Description Text (5):
In database terminology, the service model template 34 is the schema. On the other hand, an instance defines the records in the database and the values of the static information. In FIG. 2, the instance 36 is determined using auto-discovery, as will be explained fully below. Information regarding the services and service elements (e.g., servers, hosts and links) that exist in the ISP system or other service provider systems may be auto-discovered. The store representing auto-discovered information shall be referred to as the auto-discovered instance 36. In one embodiment, the auto-discovery approach is dividable into two phases. In the first phase, the information that is available in the domain name system of an ISP is used to discover the existence and the relationships among different services. In order to allow subscribers to access a host using its host name, rather than requiring specification of an IP address that is more difficult to remember, DNS is used to map the host names to the IP addresses for all hosts in the ISP system. Moreover, the exchange of email messages across hosts also occurs using the mail exchange records (MX records) maintained by the DNS. In summary, the domain name system stores a wealth of information that is used in the first phase of the auto-discovery process to discover Internet services and relationships among them. However, other mechanisms are used in this first phase to supplement the information acquired from the domain name system.

Detailed Description Text (49):
There are several active and passive measurements that can be made to assess the health of the different elements involved in supporting the Read Mail service. A measurement system (MS) may be installed in the server farm of the ISP to perform measurements using agents executing on the MS and on the different ISP hosts. The different measurements that characterize the Read Mail service include an active service quality measurement of availability and response time made from the MS in the service farm, an active measurement of network throughput from the MS in the service farm to the POP sites, passive measurements of CPU and memory utilization passive measurements of TCP connection traffic and packet traffic to the mail servers obtained from agents executing on the mail servers, and passive measurements of NFS statistics (e.g., number of calls, timeouts, and duplicate transmissions) on the mail servers and the mail content servers. The active measurements attempt to assess the service quality as viewed from subscribers connecting to the POP sites, while the passive measurements may be used to assess resource utilization and traffic statistics that are critical for problem diagnosis.

Detailed Description Text (50):
FIG. 5 is an illustration of an example of a view that may be presented to an ISP operator using the view generator 42 of FIG. 2. While the oval-shaped nodes in the service model graph represent the different services and service elements, the

arrows represent measurements of services and service elements. The root of the service model graph is the Read Mail service, represented by oval 80. The state of this node represents the overall health of the Read Mail service, as assessed by the MS located in the service farm of the ISP. That is, the overall health is assessed without considering the state of the network links from the server farm to the POP sites. In one embodiment, the overall health is represented by color coding the oval 80. For example, oval 80 may be shaded green to designate a positive health of the Read Mail service, and may be shaded red if the Read Mail service has degraded in its availability or performance.

Detailed Description Text (57):
Since services and service elements can be organized based on domains of responsibility in a service model, ISP operations personnel need only monitor and diagnose the services that fall in their domain of responsibility. In the Read Mail service example of FIG. 4, an email operations expert who is responsible for the mail service and the mail servers uses the service model depicted in FIG. 5, since the expert is mainly interested in the states of the email services and servers. The authentication and NFS services are included in the service model representation, since these services can adversely impact the Read Mail service. In contrast, the links between the service farm and the POP sites are not included in the model, since they do not affect the Read Mail service from the perspective of the email expert.

Detailed Description Text (66):
Another category of techniques may be referred to as the service-specific-but-application-independent techniques. Techniques in this category are specific to the service. They are intended to monitor, but may be used for discovery that is independent of the specific application server that is being used to implement the service. For example, the discovery of the relationship between the services offered by POP3 email servers and the service provided by SMTP-based email servers is possible using application-independent techniques, since the relationship is accessible from the domain name service in the form of mail exchange (MX) records.

Detailed Description Text (69):
An often under utilized component of an ISP system is the domain name service (DNS). In order to allow subscribers to access hosts using their host names, rather than their more difficult to remember IP addresses, DNS stores the host name-to-IP address mapping for all of the hosts in the ISP system. Moreover, the exchange of email messages across hosts occurs using the mail exchange records maintained by the DNS. Name server (NS) records in the DNS database serve to identify authoritative name service for the ISP domain--these are name servers that are externally accessible from the global Internet and are the authorities that are contacted when users in the global Internet attempt to access any hosts in the ISP system. Moreover, service groups, such as an email service group, are enabled via round-robin scheduling mechanisms implemented in the DNS servers. In summary, the domain name system holds a wealth of information that is critical for auto-discovery of ISP services. However, additional mechanisms are necessary to complement DNS-based discovery mechanisms.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L22: Entry 41 of 54 File: USPT Jun 25, 2002

DOCUMENT-IDENTIFIER: US 6412079 B1
** See image for Certificate of Correction **
TITLE: Server pool for clustered system

Abstract Text (1):
A computer system includes a plurality of interdependent processors for operating a common set of applications. Each interdependent processor executes an independent operating system image without sharing file system state information. Each interdependent processor has a network access card with a first network connection and a second network connection. The computer system includes a first active backplane coupled to each first network connection of each processor; a second active backplane coupled to each second network connection of each processor, the second active backplane operating in lieu of the first active backplane in case of a fail-over; and one or more directors coupled to the first and second active backplanes, each of the one or more directors load-balancing requests directed at a plurality of servers.

Application Filing Date (1):
19981009

Brief Summary Text (9):
A computer system includes a plurality of interdependent processors for operating a common set of applications. Each interdependent processor executes an independent operating system image without sharing file system state information. Each interdependent processor has a network access card with a first network connection and a second network connection. The computer system includes a first active backplane coupled to each first network connection of each processor; a second active backplane coupled to each second network connection of each processor, the second active backplane operating in lieu of the first active backplane in case of a fail-over; and one or more directors coupled to the first and second active backplanes, each of the one or more directors load-balancing requests directed at a plurality of servers.

Brief Summary Text (11):
In a second aspect, a method for operating a computer system includes: executing an independent operating system image without sharing file system state information by each processor in a group of interdependent processors, each interdependent processor having a network access card with a first network connection and a second network connection; transferring data on either a first active backplane coupled to each first network connection of each processor or a second active backplane coupled to each second network connection of each processor, the second active backplane operating in lieu of the first active backplane in case of a fail-over; and load-balancing requests directed at a plurality of servers using one or more directors coupled to the first and second active backplanes.

Brief Summary Text (12):
Implementations of the method include the following. The transferring step includes routing data over each active backplane using a switch, which may be an Ethernet switch. Data may be accessed from one or more networked data storage devices

connected to the first and the second active backplanes. Requests may be communicated from one or more servers over the first or the second active backplane. Each director may be connected to each of the first and second active backplanes to provide load-balancing. Each director may also be connected to a router. A peripheral device connected to the first or second active backplane may be accessed at a predetermined address. The address may be a predetermined Internet Protocol (IP) address, and the peripheral device may be accessed at the predetermined IP address from the first or second active backplane. The address may also be a predetermined Media Access Protocol (MAC) address.

Brief Summary Text (13):
Advantages of the invention include the following. The invention provides scalability and fault tolerance. The invention allows many servers to perform the same task in an active/active scalable manner. The invention also supports load balancing among a pool of like servers. By providing a client process with access to a pool of like servers which are load balanced, the invention keeps the response time for each request to a minimum. Thus, the invention supports high data availability, fast access to shared data, and low administrative costs through data consolidation. Additionally, the invention may be built using standard off-the-shelf components to reduce overall system cost.

Detailed Description Text (22):
Directors recognize a Universal Resource Locator (URL) or Internet Protocol (IP) address as being associated with a pool of servers. If a server becomes unavailable, the server request is simply put in a slow poll mode and server requests are not sent to it until it starts responding. Directors provide various load-balancing algorithms to even out the load among a pool of servers. These devices assure high availability and scalability. By using directors 216 and 220 within such a clustered system, provisioning active/active pools of servers can be provided using off the shelf components to assure scalable, load balanced, fault tolerant access of clients to all server resources.

Detailed Description Text (26):
Turning now to FIG. 7, an overview of processes executing on the computer systems described above are shown. These processes include a telephony process 300 which is a collection one or more telephony application modules and an automatic speech recognition (ASR) server module. The telephony group 300 communicates with an internal services process 320 via client/server communications. The internal services process 320 is a collection of software such as a CIFS application module, an application manager module, a facsimile manager module, a pager and message delivery manager module, a resource manager module, and an operation and administration (OA&M) manager module. The external services process 340 is a collection of services which handle tasks such as Internet Message Access Protocol 4 (IMAP4)/Post Office Protocol 3 (POP3) electronic mail, Short Message System (SMS) server, Web agent server, network and system management module, personal information management/electronic mail synchronization server module, and Web messaging server module.

Detailed Description Text (32):
FIG. 9 shows the internal services module 320 in more detail. A file sharing protocol handler 322 processes requests from client applications so that they can read and write to files or request services from server programs in the computer network. In this case, the file sharing protocol handler is a CIFS handler. By using CIFS locking strategies, system wide resources and queues can be processed locally and in parallel with peer servers such as resource managers, application managers, pager/outdialer managers, and fax managers. The lock technique improves parallelism and scalability while reducing load balancing and fail-over complexity.

Detailed Description Text (36):

The IMAP4/POP3 server 342 and the X.500 directory server 344 in turn communicates over the Internet 319. Further, the IMAP/POP3 server 342 communicates with an SMS server 346. Additionally, the X.500 directory 344 communicates with a personal information management (PIM)-electronic mail synchronization system 348. The PIM-electronic mail synchronizer 348 in turn communicates with the Internet 319 such that a user can synchronize his PIM and electronic mail over the Internet.

Detailed Description Text (39):
The above processes, in conjunction with the computer systems described above, operate in a fault-tolerant and scalable manner. Further, a pool of servers may execute these processes in an active/active scalable manner. The directors provide load balancing among the pool of like servers. By providing the above processes with access to the pool of like servers which are load balanced, the system provides fast response time for each request.

CLAIMS:

18. The method of claim 13, further comprising the step of using a director connected to both the first and second active backplanes to provide load-balancing.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L22: Entry 45 of 54 File: USPT Dec 4, 2001

DOCUMENT-IDENTIFIER: US 6327622 B1
TITLE: Load balancing in a network environment

Abstract Text (1):
A method is provided for load balancing requests for an application among a plurality of instances of the application operating on a plurality of servers. A policy is selected for choosing a preferred server from the plurality of servers according to a specified status or operational characteristic of the application instances, such as the least-loaded instance or the instance with the fastest response time. The policy is encapsulated within multiple levels of objects or modules that are distributed among the servers offering the application and a central server that receives requests for the application. A first type of object, a status object, gathers or retrieves application-specific information concerning the specified status or operational characteristic of an instance of the application. Status objects interact with instances of the load-balanced application and are configured to store their collected information for retrieval by individual server monitor objects. An individual server monitor object illustratively operates for each server operating an instance of the application and retrieves the application-specific information from one or more status objects. A central replicated monitor object gathers the information from the individual server monitor objects. The information is then analyzed to select the server having the optimal status or operational characteristic. An update object updates the central server, such as a domain name server, to indicate the preferred server. Requests for the application are then directed to the preferred server until a different preferred server is identified.

Application Filing Date (1):
19980903

Parent Case Text (1):
U.S. Pat. No. 6,092,178, entitled "Systems for Responding to a Resource Request," and U.S. patent application Ser. No. 09/146,848, entitled "Load Balancing for Replicated Services," both of which were filed on Sep. 3, 1998, are related to the present application.

Brief Summary Text (2):
This invention relates to the field of computer systems. More particularly, a system and methods are provided for load balancing among application programs or replicated services.

Brief Summary Text (4):
A service offered simultaneously on multiple servers is often termed "replicated" in recognition of the fact that each instance of the service operates in substantially the same manner and provides substantially the same functionality as the others. The multiple servers may, however, be situated in various locations and serve different clients. Application programs may also operate simultaneously on multiple servers, with each instance of an application operating independently of, or in concert with, the others. In order to make effective use of an application or replicated service offered by multiple servers (e.g., to satisfy clients'

requests), there must be a method of distributing clients' requests among the servers and/or among the instances of the application or service. This process is often known as load balancing. Methods of load balancing among instances of a replicated service have been developed, but are unsatisfactory for various reasons.

Brief Summary Text (5):
In one method of load balancing a replicated service, clients' requests are assigned to the servers offering the service on a round-robin basis. In other words, client requests are routed to the servers in a rotational order. Each instance of the replicated service may thus receive substantially the same number of requests as the other instances. Unfortunately, this scheme can be very inefficient.

Brief Summary Text (7):
In another method of load balancing, specialized hardware is employed to store information concerning the servers hosting instances of a replicated service. In particular, according to this method information is stored on a computer system other than the system that initially receives clients' requests. The stored information helps identify the server having the smallest load (e.g., fewest client requests). Based on that information, a user's request is routed to the least-loaded server. In a web-browsing environment, for example, when a user's service access request (e.g., a connection request to a particular Uniform Resource Locator (URL) or virtual server name) is received by a server offering Domain Name Services (DNS), the DNS server queries or passes the request to the specialized hardware. Based on the stored information, the user's request is then forwarded to the least-loaded server offering the requested service.

Brief Summary Text (10):
As mentioned above, present load balancing techniques are also limited in scope. For example, the techniques described above are designed for replicated services only and, in addition, only consider the operational status or characteristics of the servers hosting the replicated service, not the service itself. In other words, present techniques do not allow load balancing among instances of an application program or, more generally, the collection or consideration of information concerning the status of individual instances of applications or services executing on multiple servers.

Brief Summary Text (13):
A load balancing policy is selected for distributing the client requests among the multiple servers and instances of the application and, at periodic intervals, a "preferred" server is identified in accordance with the policy. Illustratively, the selected policy reflects or specifies one or more application-specific factors or characteristics to be considered in choosing the preferred server. Client requests are routed to the preferred server until such time as a different server is preferred. A selected load balancing policy may be replaced while the application continues operating.

Drawing Description Text (2):
FIG. 1 is a block diagram depicting an illustrative environment in which an embodiment of the present invention may be implemented to load balance client requests among multiple instances of an application.

Drawing Description Text (5):
FIG. 4 is a flow chart demonstrating the generation of objects in a load-balancing framework in accordance with an embodiment of the present invention.

Drawing Description Text (6):
FIG. 5 is a flow chart demonstrating the registration of objects within a load balancing framework and their use in monitoring an instance of a load-balanced

application in accordance with an embodiment of the present invention.

Detailed Description Text (3):
In particular, illustrative embodiments of the invention are described in the context of applications such as a database management system (DBMS), electronic mail, or web browsing. Various embodiments of the invention may therefore involve the use of a central server, such as a Domain Name Services (DNS) server, to resolve an access request for an application into an address of a physical machine such as a computer server. One skilled in the art will recognize that the present invention is not limited to the applications described herein or the use of a DNS server, and may be readily adapted to other applications and services for which load balancing is appropriate.

Detailed Description Text (9):
The specific information that is collected (from the various application instances and, possibly, the host servers) is determined by a load balancing policy that may be selected by a system manager or administrator. The preferred server is then selected by analyzing the collected information. Thus, in one illustrative policy, the preferred server is the server offering the application instance that is least-loaded (e.g., has the fewest pending client requests or fewest connected clients). In another illustrative policy, the preferred server is the server closest to the central server.

Detailed Description Text (11):
In one embodiment of the invention a standard application programming interface (API) is provided to construct and apply the load balancing framework described below. With the standard API, a programmer may generate application-specific status objects (described in detail below in conjunction with FIG. 2) which, when executed, gather the information described above. The application-specific status objects may, in addition, interact with the application in accordance with an application-specific API.

Detailed Description Text (12):
Generating application-specific status objects or modules illustratively allows the collection of any information that could form the basis for load balancing client requests. For example, to load-balance a database application, it may be desirable to determine the number of users being serviced by each instance of the application, the number of users that have accessed an instance, or the number of access requests that are pending with or that have been processed by each instance. The information gathered by the application-specific status objects is used by other objects and/or modules in the load-balancing framework in order to determine a preferred server.

Detailed Description Text (16):
In the environment of FIG. 1, when client 120 attempts to connect to application 104, the access request is received by central server 100. Central server 100, through lookup table 102, identifies a preferred server offering an instance of program 104 and routes the client request accordingly. The server identified in lookup table 102 may be determined according to a load-balancing policy, as discussed below. Further, the server identified in lookup table 102 is updated or changed from time to time in accordance with the selected policy in order to distribute client requests among the instances of the application.

Detailed Description Text (17):
In a present embodiment of the invention, information reflecting the status or operation of application instances 104a, 104b and 104c (and/or servers 110, 112 and 114) is collected and analyzed on a regular or periodic basis. The information that is collected is identified in a load balancing policy that identifies one or more factors or pieces of information to be used to identify a "preferred" server to which client requests for application 104 are to be routed. Different policies thus

require different information to be collected from the application instances, and the active policy can be changed during load balancing.

Detailed Description Text (19):
In one embodiment of the invention, status objects are generated or produced to collect application-specific data from the application instances. The status objects may be constructed according to a standard API for a present load-balancing framework. Status objects and the load-balancing framework are described in detail below with reference to FIG. 2. In one particular embodiment, status objects (and other objects within the load-balancing framework) are designed (e.g., an object class is constructed) according to the standard API in a generation stage. Then, in a registration stage, individual objects are instantiated from the class(es). Finally, in a monitoring stage, the objects begin collecting information.

Detailed Description Text (21):
Besides status objects, other computer-readable instructions (e.g., in the form of objects, agents or modules) are also executed (also described below) to collect, assemble and analyze the various pieces of information provided by the status objects and to update lookup table 102. The objects or agents within a load balancing framework may be created in a suitable programming or script language and then configured and installed on each of servers 110, 112 and 114 and/or on central server 100.

Detailed Description Text (34):
In summary, when load balancing is performed in accordance with the embodiments of the invention described above, a status object gathers load and/or operational information for an instance of the application being load-balanced. An IMO interfaces with or otherwise retrieves the information from each status object and an RMO gathers the information from all application instances from the IMOs.

Detailed Description Text (38):
The status objects, IMOs, RMO and updater object may be considered to comprise a load-balancing framework for distributing client requests among various instances of an application. As one skilled in the art will recognize, the different objects within the framework may be distributed among the servers hosting application instances, a central server, and other entities such as intermediate servers.

Detailed Description Text (40):
Each server farm in the presently described embodiment includes an intermediate server (i.e., server 306 in server farm 300 and server 316 in server farm 310). One function of an intermediate server in this embodiment is to collect, from the servers in the farm that host instances of the load-balanced application information necessary to select a preferred server. For example, intermediate replicated monitor object (IRMO) 306a is operated on intermediate server 306 to collect data from servers 302 and 304. IRMO 306a operates similarly to the RMO described above in conjunction with to FIG. 2, but in this embodiment is located on a server situated between central server 100 and the servers offering the application. The load balancing framework of the illustrated embodiment also includes status objects (e.g., depicted by numerals 302a, 304a, 312a and 314a) and IMOs (e.g., depicted by numerals 302b, 304b, 312b and 314b) operating on servers 302, 304, 312 and 314.

Detailed Description Text (43):
In another alternative embodiment of the invention, load balancing among instances of an application is performed among multiple participating servers wherein one or more of the servers are segregated (e.g., situated in a remote location and/or within a server farm). Within the group of segregated servers, a "local" load balancing policy may be implemented for distributing all client requests sent to the group and/or to a specific member of the group. In this alternative embodiment, the segregated servers may be considered a single entity for the purposes of a

"global" load balancing policy specifying the manner in which client requests for the application are to be distributed among participating servers. The global and local policies need not be equivalent (e.g., the global policy may require selection of the closest server (or group of servers) while the local policy may require the least-loaded server or application instance).

Detailed Description Text (44):
With reference now to FIGS. 4 and 5, an illustrative method of load balancing between multiple instances of an application is depicted. In the illustrated method, a central server (e.g., a DNS Server) resolves client requests for a virtual name by which the application is known into an identifier of a preferred server offering an instance of the application. Each instance of the application illustratively operates on a separate server and is modified to produce application-specific information needed to choose the preferred server.

Detailed Description Text (45):
FIG. 4 demonstrates an illustrative generation stage of the method, in which objects in the load-balancing framework are designed (e.g., object classes are constructed). FIG. 5 demonstrates illustrative registration and monitoring stages, in which individual objects are created (e.g., instantiated) and begin collecting information from instances of the load-balanced application.

Detailed Description Text (46):
With reference now to FIG. 4, state 400 is a start state. In state 402 a policy to be applied to identify a preferred server is selected. One skilled in the art will appreciate that various policies are possible, depending upon the nature of the application and the aspect(s) of the application that are conducive to load balancing.

Detailed Description Text (50):
In state 404, sequences of instructions or executable code are produced for performing the function(s) of the status objects (i.e., to collect the application-specific information needed to choose a preferred server). In one embodiment of the invention in which the load balancing framework is constructed using an object-oriented programming language, a compatible language and basic building blocks provided by the framework are used to generate the status objects, IMOs, RMO and specialized object. Thus, in this embodiment of the invention state 404 comprises the creation of one or more classes of status objects, from which individual instances will be created in the registration stage depicted in FIG. 5. Illustratively, status objects are substantially similar for each instance of a load-balanced application.

Detailed Description Text (51):
Status objects may be configured to store the information for retrieval by individual monitor objects or, alternatively, to interface with the IMOs directly in order to pass the information along. In addition, the status objects may be configured to execute automatically on a regular basis, in response to action by another part of the load balancing framework (e.g., upon invocation by an IMO), the application or some other external entity, etc.

Detailed Description Text (52):
As discussed above, in a current embodiment of the invention status objects (and other framework objects) are constructed using an object-oriented programming language. One skilled in the art will recognize that many suitable programming languages and tools exist and that the invention may be implemented using techniques other than object-oriented programming. Illustratively, however, status objects substantially adhere to a common format (e.g., detailed in a load balancing framework API) in order to cooperate with the overall load balancing framework.

Detailed Description Text (53):

In state 406, the existing load-balancing framework is examined to determine whether an IMO (e.g., an IMO class) already exists for collecting data concerning an instance of the load-balanced application. If an IMO already exists, the illustrated method continues at state 410. Otherwise, in state 408 an IMO structure (e.g., an object class) is constructed that is specific to the application instance. The IMO is designed such that it will collect the various data and statistics gathered by one or more status object(s). In an alternative embodiment of the invention, the IMOs generated for all instances of a particular application are substantially similar.

Detailed Description Text (54):
In state 410, the existing load balancing framework is examined to determine whether an RMO already exists for receiving data from the IMOs that are associated with each instance of the application. As described above, in one embodiment of the invention an RMO comprises a data structure for retaining application-specific information from the application instances. If an RMO already exists, the illustrated method continues at state 414. Otherwise, in state 412, an RMO structure (e.g., an object class) is constructed that is specific to the application. As with the status objects and IMOs, an actual RMO instance will be created as part of the registration stage depicted in FIG. 5.

Detailed Description Text (55):
In state 414, the existing load balancing framework is examined once more. This time, it is determined if the sequence of instructions or executable code for the specialized object that will determine a preferred server already exists. If not, in state 416 a specialized object structure (e.g., an object class) is constructed to apply the selected load balancing policy to the results of the data collected concerning the various application instances (and/or their host servers) and select a preferred server. The specialized object is also designed to update the lookup table (or other data structure) to store an identity of the preferred server.

Detailed Description Text (57):
With reference now to FIG. 5, illustrative registration and monitoring stages of the illustrated method are depicted. For present purposes, the term registration refers to the registration of individual objects (e.g., status object, IMO, RMO, specialized object) within a load balancing framework, including their creation (e.g., instantiation) from the object structures (e.g., classes) produced in the generation stage depicted in FIG. 4. In the monitoring stage, information is collected for the purpose of identifying a preferred server in accordance with a selected load balancing policy. In FIG. 5, state 500 is a start state.

Detailed Description Text (58):
In state 502, a status object is registered with the load-balancing framework. In one embodiment of the invention, the standard API provided with the load balancing framework includes a command (e.g., "create") for creating an instance of each object within the framework. As one skilled in the art will appreciate, creating an instance of an object, such as a status object, involves the dynamic loading and executing of a sequence of instructions defining the object.

Detailed Description Text (60):
In state 506, an individual monitor object (IMO) is registered with the load-balancing framework. Illustratively, one IMO is registered or created for each instance of the application. Each IMO may be installed on the server executing the associated instance of the application. In an alternative embodiment, however, IMOs operate on the central server or an intermediate server located between the central server and the host servers. As described above, IMOs may be configured to collect and report certain information or data. In the presently described embodiment of the invention, the collected information is received directly from a status object. In an alternate embodiment of the invention, the information may be retrieved from a location in which it was placed by the status object (e.g., a storage device, a

file or other data structure).

Detailed Description Text (61):
As described above, the information to be collected may be determined by the selected load balancing policy, and will be used to identify a preferred server. In a present embodiment of the invention, the active policy for an application may be changed without disrupting the handling of client requests. Illustratively, this is done by temporarily pausing the operation of IMOs for the application, installing new status objects reflecting the new policy, then resuming the IMOs. Advantageously, the IMOs need not be altered or replaced.

Detailed Description Text (64):
A back-end or host server (e.g., server 110 from FIG. 1) may be removed from or added to the load-balancing scheme without significantly disrupting operation of the application. A host server may, for example, become inoperative or require replacement. Illustratively, each RMO maintains an index (e.g., in an array, linked list, vector, other data structure, etc.) of all servers participating in the load balancing (e.g., all servers offering an instance of the application). This information may, for example, be included in a list of IMOs from which the RMO receives information. By temporarily pausing the RMO, removing the IMO associated with the server from the list and restarting the RMO, the RMO will stop attempting to retrieve information for the removed server (i.e., the RMO will no longer communicate with the IMO associated with the server). Servers may be added to the load-balancing scheme in a similar manner.

Detailed Description Text (65):
In state 514, a specialized object is registered with the load-balancing framework (e.g., created from its object class). In state 516, parameters concerning the operation of the specialized object are set. Illustrative parameters include an identity of the RMO, the frequency of information retrieval from the RMO, an identity of the lookup table, method of interfacing with the RMO and/or lookup table, etc. In one embodiment of the invention, the specialized object analyzes the information collected from the servers hosting the application instances, identifies a preferred server in accordance with the load-balancing framework and updates the lookup table.

Detailed Description Text (66):
Where, for example, the application comprises web browsing on web servers, the specialized object may take the form of a DNS updater configured on a DNS server to modify a DNS zone file to identify the server to which requests are to be routed. Similarly, where load balancing is being performed for an application operating in a master/slave relationship (e.g., a master process or server routes requests to slave processes or servers), the specialized object updates a data structure or entry indicating a preferred process or server.

Detailed Description Text (72):
In one alternative embodiment of the invention, for example, clients access an instance of the application program directly (i.e., rather than connecting through a central server). In this alternative embodiment, the program instances exchange information (e.g., via status objects and/or other elements of a load-balancing framework) and redirect client requests as necessary to balance the requests in accordance with the selected policy.

Detailed Description Text (73):
In another alternative embodiment of the invention, one or more elements of a load-balancing framework are combined. By way of illustration, an RMO may be designed to perform the functions of an IMO and collect information from one or more status objects.

Other Reference Publication (1):

Pending U.S. Patent Application Ser. No. 09/146,848, by Anita Jindal, et al., titled "Load Balancing for Replicated Services," filed Sep. 3, 1998, with Attorney Docket No. SUN-P3316-JTF.

CLAIMS:

14. A method of load balancing requests for an application received at a central server among a set of servers, wherein each server in the set of servers operates an instance of the application, comprising:

selecting a policy for directing a request for the application to a preferred server, wherein said policy reflects a server factor for selecting said preferred server from the set of servers;

configuring a first status object to determine a first status of said server factor for a first instance of the application;

configuring a first server monitor object to receive said first status;

configuring a central monitor object to receive multiple statuses of said server factor for multiple instances of the application, including said first status;

examining said multiple statuses to select a preferred server; and

updating the central server to identify said preferred server;

wherein said server factor comprises a performance indicator specific to the application.

36. An apparatus for load balancing requests for an application received at a central server, comprising:

a first status determination means for determining a first application-specific status of a first instance of the application;

a second status determination means for determining a second application-specific status of a second instance of the application;

central monitor means for receiving said first application-specific status and said second application-specific status;

server selection means for selecting a preferred server from one of said first server and said second server; and

updating means for storing an identifier of said preferred server on the central server.

39. A method of load-balancing multiple requests for an application, wherein instances of the application execute on a plurality of servers, the method comprising:

receiving a client request for an application at a first server operating a first instance of the application;

executing a first status module on the first server, wherein said first status module is configured to determine a first status of the first instance;

executing a server monitor module on the first server, wherein said server monitor module is configured to receive a first status of a second instance of the application operating on a second server;

examining said first status of the first instance and said first status of the second instance to select a preferred server from among the plurality of servers; and

routing the client request to said preferred server.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L22: Entry 48 of 54 File: USPT Jun 5, 2001

DOCUMENT-IDENTIFIER: US 6243676 B1
TITLE: Searching and retrieving multimedia information

Application Filing Date (1):
19981223

Brief Summary Text (16):
The invention also provides scalability and fault tolerance in delivering the value-added information. The invention allows many servers to perform the same task in an active/active scalable manner. The invention also supports load balancing among a pool of like servers. By providing a client process with access to a pool of like servers which are load balanced, the invention keeps the response time for each request to a minimum. Thus, the invention supports high data availability, fast access to shared data, and low administrative costs through data consolidation. Additionally, the invention may be built using standard off-the-shelf components to reduce overall system cost.

Detailed Description Text (62):
Directors recognize a Universal Resource Locator (URL) or Internet Protocol (IP) address as being associated with a pool of servers. If a server becomes unavailable, the server request is simply put in a slow poll mode and server requests are not sent to it until it starts responding. Directors provide various load-balancing algorithms to even out the load among a pool of servers. These devices assure high availability and scalability. By using directors 216 and 220 within such a clustered system, provisioning active/active pools of servers can be provided using off the shelf components to assure scalable, load balanced, fault tolerant access of clients to all server resources.

Detailed Description Text (69):
The telephony process 300 communicates with an internal services process 320 via client/server communications. In the internal services module 320, a file sharing protocol handler processes requests from client applications so that they can read and write to files or request services from server programs in the computer network. In this case, the file sharing protocol handler is a CIFS handler. By using CIFS locking strategies, system wide resources and queues can be processed locally and in parallel with peer servers such as resource managers, application managers, pager/outdialer managers, and fax managers. The lock technique improves parallelism and scalability while reducing load balancing and fail-over complexity.

Detailed Description Text (72):
The internal services process 320 is a collection of software such as a CIFS application module, an application manager module, a facsimile manager module, a pager and message delivery manager module, a resource manager module, and an operation and administration (OA&M) manager module. The external services process 340 is a collection of services which handle tasks such as Internet Message Access Protocol 4 (IMAP4)/Post Office Protocol 3 (POP3) electronic mail, Short Message System (SMS) server, Web agent server, network and system management module, personal information management/electronic mail synchronization server module, and

Web Messaging server module.

Detailed Description Text (76):
The IMAP4/POP3 server and the X.500 directory server in turn communicates over the Internet. Further, the IMAP/POP3 server communicates with an SMS server. Additionally, the X.500 directory communicates with a personal information management (PIM)-electronic mail synchronization system. The PIM-electronic mail synchronizer in turn communicates with the Internet such that a user can synchronize his PIM and electronic mail over the Internet.

Detailed Description Text (79):
The above processes, in conjunction with the computer systems described above, operate in a fault-tolerant and scalable manner. Further, a pool of servers may execute these processes in an active/active scalable manner. The directors provide load balancing among the pool of like servers. By providing the above processes with access to the pool of like servers which are load balanced, the system provides fast response time for each request.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L22: Entry 50 of 54 File: USPT May 8, 2001

DOCUMENT-IDENTIFIER: US 6230164 B1
TITLE: Communication system with rapid database synchronization

Application Filing Date (1):
19980505

Drawing Description Text (18):
FIG. 13 is an exemplary flowchart of a load balance request process;

Drawing Description Text (19):
FIG. 14 is an exemplary flowchart of a load balancing process;

Detailed Description Text (4):
AIN (Advanced Intelligent Network) system 22 includes one or more STPs (signal transfer points) 24 coupled to the PSTN 12. The STPs 24 are coupled to one another and to a plurality of SCP (service control point) pairs 26. Each SCP pair 26 comprises two fully redundant SCPs 26a and 26b, which are described in greater detail herein below. STPs 24 are also connected to an NCC (network control center) 28, an SMS (service management system) 30 and VMS (voice mail system) 32. NCC 28, SMS 30 and VMS 32 are coupled to SCP pairs 26. NCC 28 includes a CGTT (Centralized Global Title Table) 34.

Detailed Description Text (27):
It is important to note that the default or initial file assignment may be modified subsequently depending on load and traffic conditions. Each IPU maintains statistics on the number of queries it receives and reports the statistics. The file assignments may then be redistributed so that any IPU is not overworked. Details of load balancing to achieve a more even distribution is described below.

Detailed Description Text (28):
Accordingly, PM Database Manager 52 is primarily responsible for database load balancing of the IPUs in SCP 30, and APG Database Manager 54 is primarily responsible for the management of the database loads on IPUs in the respective APG. The IPUs have at least three service states: IN_SERVICE, OS_MIN, and OUT_OF_SERVICE. PM Database Manager 52, APG Database Manager 54, and IPU Database Manager 66-70 may coordinate to unmount filesystems from OS_MIN and OUT_OF_SERVICE IPUs and redistribute the filesystems to the remaining IN-SERVICE IPUs. Files may also be moved among filesystems to evenly distribute the load carried by each IPU and APG. Details on the operating states of the processes are discussed in the co-pending U.S. patent application, Ser. No. 08/526,953, titled System and Method for Multi-Site Distributed Object Management Environment, which is incorporated by reference herein.

Detailed Description Text (36):
The default IPU is the IPU that the filesystem was initially assigned to; the current IPU is the IPU that presently mounted the filesystem as affected by database reconfiguration and/or load balancing. IPU table 92 maintains information for each IPU in the system, and may include:

Detailed Description Text (46):
PM Database Manager process 52 includes a number of objects to perform the task of managing the database. A short description follows, but more detailed discussion of the function of these objects are set forth below in conjunction with references to FIGS. 7-16. As shown in FIG. 4, PM Database Handler 96 performs load balancing among the IPUs, and for handling solicited requests from the host for routing information. Route Table Access 100 and Database Config Table Access 102 are objects residing in PM Database Manager 52 that control access to route table 94 and database configuration table 90, respectively. Load Balance Handler 104 is an object that contains the processing methods for load balancing files and filesystems. Shared Memory Array 106 is an array of Boolean values in shared memory 72-76 (FIG. 3) which is used to synchronize load balancing and reconfiguration between PM Database Manager 52 and APG Database Manager 54.

Detailed Description Text (57):
It has been noted above that database loads may be balanced among the IPUs in an APG so that an even distribution of query traffic is achieved. Further, because IPUs may fail or enter into one of the non-operational states (OS_MIN or OUT_OF_SERVICE), the database loads may need to be reconfigured or redistributed to the remaining IN_SERVICE IPUs. In order to synchronize load balancing and database reconfiguration between PM Database Manager 52 and APG Database Managers 54, instances of Shared Memory Array 120 are instantiated, one is Reconfig Array, an array of booleans in shared memory, and the other is Load Balance Flag, a Boolean flag also maintained in shared memory. If the database in a particular APG is being reconfigured due to one or more IPUs going down or re-entering service, the appropriate APG Database Manager 54 sets its corresponding flag in Reconfig Array. Once database reconfiguration is completed, APG Database Manager 54 resets its flag in Reconfig Array. Similarly, while load balancing is being performed, the Load Balance Flag is set by PM Database Manager 52.

Detailed Description Text (58):
FIGS. 13-15 are flowcharts demonstrating the process to synchronize load balancing and database reconfiguration. In FIG. 13, an exemplary load balance request process 320 is shown. A load balance may be requested by craft persons through a craft screen interface, by PM Database Manager 52, or by APG Database Manager 54. The Reconfig Array is first checked to see whether the Reconfig Flag is set for the APG in question, as shown in block 322. If the Reconfig Flag is set, then load balancing is simply aborted in block 324 and may be re-attempted at a later time. Because load balancing is not a critical operation, it is not required that load balancing waits for reconfiguration to complete, although such mechanisms may be instituted. If the Reconfig Flag for the APG is not set, then the Load Balance Flag is set, as shown in block 326, and load balancing is performed, as shown in block 328.

Detailed Description Text (59):
Load balancing is shown in exemplary flowchart in FIG. 14, beginning in block 340. A request to move one or more specific filesystems to one or more specific IPU is received, as shown in block 342. The request is likely to be generated by a craft person, or PM or APG Database Manager in view of the current load distribution and traffic conditions. In block 344, Database Route Control 112 makes the necessary changes to the tables to reflect the balanced load distribution. The new database loads are provided to both source and destination IPUs by PM Database Handler 96, as shown in block 346. If at this time it is detected that the source and/or destination IPU failed, as shown in block 348, load balancing is simply terminated in block 354. Otherwise, Database Route Control 98 extracts the new routing information from route table 94 and provides it to the host, as shown in blocks 350 and 352.

Detailed Description Text (60):
FIG. 15 shows the process flow for beginning database reconfiguration, beginning in

block 360. If database reconfiguration is desired, the appropriate Reconfig Flag is set for the APG, as shown in block 362. Next, a retry counter or timer (RETRY_CNT) is reset to zero, as shown in block 364. Execution then enters a loop in which the reconfiguration process waits for load balancing to complete if it is in progress. The retry counter is first checked to see if it has reached a predetermined upper limit, for example 180, as shown in block 368. If the upper limit has been reached, it is determined that the PM node has failed and its status is downgraded to the OS_MIN state. If the retry count has not yet reached the predetermined upper limit, then the Load Balance Flag is checked to see if it is set, as shown in block 370. If it is not set, then execution may proceed with database reconfiguration. Otherwise, the retry counter is incremented and a predetermined amount of time, for example one second, is permitted to elapse before returning to the beginning of the loop at block 366.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L27: Entry 18 of 31 File: USPT Jun 26, 2001

DOCUMENT-IDENTIFIER: US 6250930 B1
TITLE: Multi-functional communication and aggregation platform

Application Filing Date (1):
19980529

Drawing Description Text (5):
FIG. 3 illustrates the inter-user communication flow in an exemplary current distributed e-mail system;

Detailed Description Text (45):
Current Distributed E-Mail Systems: In present day distributed e-mail systems, a user sends a message (or a copy of the message) simultaneously to a group of on-line users. The message recipients may or may not respond to the original message. This is illustrated in FIG. 3 where a user A broadcasts a message 301 to users B, C & D simultaneously by sending a single message to a mail clerk 350 who makes three copies of the original message and delivers those to B, C & D as shown at 311, 312 & 313 respectively.

Detailed Description Text (48):
In contrast to the above, in the e-Logic Aggregation Engine, a user sends a multimedia message to a number of recipients. Responses are aggregated and analyzed for the user without requiring each e-mail response to be separately accessed or opened. Reports, graphs, charts and views of the data as well as the database itself can be shared with other users. The e-Logic system thus allows for one-to-many and many-to-one multimedia communication and analysis. This is shown in FIG. 6 where a user A composes an multimedia enhanced e-mail message 601 and transfers it to a Mail Distributor 650 which replicates the message as necessary and transmits these to a set of recipients B, C, D, E & F as shown at 611-615. Some or all of the recipients review the received message and respond back to the originator or a designated response collector as shown at 621-625.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L27: Entry 20 of 31 File: USPT Mar 6, 2001

DOCUMENT-IDENTIFIER: US 6199062 B1
TITLE: Reverse string indexing in a relational database for wildcard searching

Application Filing Date (1):
19981119

Brief Summary Text (7):
To this end, the Lightweight Directory Access Protocol (LDAP) has emerged as an IETF open standard to provide directory services to applications ranging from e-mail systems to distributed system management tools. LDAP is an evolving protocol that is based on a client-server model in which a client makes a TCP/IP connection to an LDAP server, sends requests, and receives responses. The LDAP information model in particular is based on an "entry," which contains information about some object. Entries are typically organized in a specified tree structure, and each entry is composed of attributes.

Brief Summary Text (19):
These and other objects of the invention are described in a method for wildcard searching a relational database, e.g., by using hierarchical, filter-based queries. The method begins by generating a forward index of the character strings in the database. The method then generates a reverse index of the character strings in the relational database. The reverse index is generating, preferably by mirroring (i.e. creating a mirror image of) the forward index. Upon receipt of a hierarchical, filter-based query having a search string with a wildcard, the reverse index is then used to generate the corresponding relational database query (e.g., in SQL) if the wildcard is at a given position within the search string. Thus, for example, the given position is a leading position in the search string. Alternatively, the given position is an intermediate position in the search string and a number of characters trailing the wildcard is greater than a number of characters leading the wildcard. If the wildcard is not at the given position within the search string, the forward index is used to generate the relational database query. In either case, the relational database query is then used to access the relational database and return search results.

Detailed Description Text (11):
The process for generating the reverse character strings is now described. As noted above, LDAP entries consist of attribute-value pairs. Within the relational database (e.g., DB/2), a separate table is created for each of the attributes allowed within the LDAP schema. Within this table, the attribute values are stored, along with an entry ID (EID) to which the attribute values belong. If the value of the attribute is less than the value for an indexable column, then an index is created based on the attribute value and the EID. Further, if the value exceeds the maximum length for an indexable column, an additional column is created that contains a truncated value. This column with the truncated value is then indexed based on the value in the EID. This index is referred to herein as an forward index. According to the invention, the forward index is mirrored into a reverse index to facilitate wildcard searching. In particular, within the LDAP schema files, a given user (e.g., the directory administrator) provides attribute names and characteristics. The table below illustrates several attribute definitions

provided as a default schema:

Detailed Description Text (16):
According to the present invention, to improve searches that begin with (or that otherwise include) a wildcard, an additional column is added. Preferably, the contents of this column are the mirror image of those within the forward-indexed column. Thus, for example, for table CN as described above, the forward-indexed column is the cn column. For the MEMBER table example, the forward-indexed column is the MEMBER_T column. As will be seen, the resulting data generate a so-called reverse index that is used to facilitate the wildcard search strategies described above and illustrated in FIGS. 6-8.

CLAIMS:

6. The method as described in claim 1 wherein the reverse index is generated by creating a mirror image of a forward index.

8. A method for wildcard searching a relational database using hierarchical, filter-based queries, comprising the steps of:

generating a forward index of character strings in the relational database;

generating a reverse index by mirroring the forward index;

upon receipt of a hierarchical, filter-based query having a search string with at least one wildcard, determining whether the forward index, the reverse index, or both indices, should be used to generate a relational database query;

as a result of the determination, generating the relational database query; and

using the relational database query to access the relational database.

13. A method for searching a relational database from a Lightweight Directory Access Protocol (LDAP) directory service generating filter-based queries, the relational database having stored therein a forward index of character strings and a reverse index that is a mirror image of the forward index, the method comprising the steps of:

upon receipt of an LDAP query having a search string with a wildcard, using the forward index, the reverse index, or both indices, to generate a relational database query depending on a position of at least one wildcard within the search string; and

using the relational database query to access the relational database.

18. A computer program product in computer-readable media for searching a relational database using hierarchical, filter-based queries, comprising:

means for generating a reverse index by mirroring a forward index of character strings in the relational database; and

means responsive to receipt of a hierarchical, filter-based query having a search string with at least one wildcard, for using the forward index, the reverse index, or both indices, to generate a relational database query.

19. A directory service, comprising:

a directory organized as a naming hierarchy having a plurality of entries each represented by a unique identifier;

a relational database management system having a backing store for storing directory data;

means for searching the directory, comprising:

means for generating a reverse index by mirroring a forward index of character strings in the relational database; and

means responsive to receipt of a hierarchical, filter-based query having a search string with at least one wildcard, for using the forward index, the reverse index, or both indices, to generate a relational database query.

21. In a directory service having a directory organized as a naming hierarchy, the hierarchy including a plurality of entries each represented by a unique identifier, the improvement comprising:

a relational database management system having a backing store for storing directory data;

means for searching the directory, comprising:

means for generating a reverse index by mirroring a forward index of character strings in the relational database; and

means responsive to receipt of a hierarchical, filter-based query having a search string with at least one wildcard, for using the forward index, the reverse index, or both indices, to generate a relational database query.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L39: Entry 2 of 14 File: USPT Sep 17, 2002

DOCUMENT-IDENTIFIER: US 6453404 B1
** See image for Certificate of Correction **
TITLE: Distributed data cache with memory allocation model

Application Filing Date (1):
19990527

Detailed Description Text (102):
If the requested data item is in the identified remote cache, the result of decision 328 is YES. In the example presented above, the initial hash process of step 308 (see FIG. 12) indicated that the requested data item was in the remote cache 4 and the hash process of step 322 indicated that the requested data item is in the remote cache 3. In step 330, the system retrieves the requested data item from the identified remote cache (e.g., the remote cache 3) and routes the requested data item in accordance with the flags accompanying the Get( ) command. In step 332, the system 200 executes a Put( ) command to store the requested data item in the remote cache that was identified by the routing data from the new map in step 308 (see FIG. 12) and the process ends at step 334. In the example discussed above, the remote cache 4 was initially identified by the hash process of step 308 as the location of the requested data item. Thus, in step 332, the Put( ) command stores the requested data item in the remote cache 4. In this manner, the system has begun to balance the stored data items between all of the identified remote caches (e.g, the remote caches 1-4). As additional Get( ) commands are executed by the system, some requested data items stored in the remote caches 1-3 will be transferred to the remote cache 4 until the system 100 achieves some load balance between the remote caches 1-4. If the new remote cache (e.g., the remote cache 4) has a different storage capacity than the other remote caches (e.g., the remote caches 1-3), the mapping process described herein can be modified to balance the workload based on the relative storage capacities of the various remote caches. Such alteration to the mapping process can be accomplished by one skilled in the art without additional information. Thus, the system 100 advantageously allows new remote caches to be added to the system on the fly and achieves load balancing between the remote caches to accommodate the addition of the new remote cache(s).

Detailed Description Text (111):
The process of retrieving data from remote caches at different levels, such as the remote caches 1-9 illustrated in FIG. 10, is illustrated in the flowchart of FIG. 14. At a start 350, it is assumed that all remote caches have been initialized and are operational. For the sake of clarity, it is further assumed that there are no newly added remote caches that require load balancing, which is illustrated in the flowcharts of FIGS. 12 and 13. In step 352, the client 102 (see FIG. 5) generates a Get( ) command to retrieve a requested data item. As noted above, the Get( ) command includes a key to uniquely identify the requested data item.

Detailed Description Text (136):
Similarly, the lists B1-L1 of FIG. 18 each independently replicate the functionality of the list A1. That is, each list independently performs the function of the list A1 as described above with respect to FIGS. 16 and 17. In addition, each of the lists B1-L1 may also include multiple lists, such as lists

B2-L2 (not shown). It should be noted that the lists A1-L1 need not have the same number of higher priority utilization lists associated therewith. For example, the list A1 may have associated lists A2-A3, while the list B1 may only have an associated list B2. The system 100 can accommodate any variable number of lists associated with the lists A1-L1.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L39: Entry 3 of 14 File: USPT Sep 10, 2002

DOCUMENT-IDENTIFIER: US 6449695 B1
** See image for Certificate of Correction **
TITLE: Data cache using plural lists to indicate sequence of data storage

Application Filing Date (1):
19990527

Detailed Description Text (102):
If the requested data item is in the identified remote cache, the result of decision 328 is YES. In the example presented above, the initial hash process of step 308 (see FIG. 12) indicated that the requested data item was in the remote cache 4 and the hash process of step 322 indicated that the requested data item is in the remote cache 3. In step 330, the system retrieves the requested data item from the identified remote cache (e.g., the remote cache 3) and routes the requested data item in accordance with the flags accompanying the Get( ) command. In step 332, the system 200 executes a Put( ) command to store the requested data item in the remote cache that was identified by the routing data from the new map in step 308 (see FIG. 12) and the process ends at step 334. In the example discussed above, the remote cache 4 was initially identified by the hash process of step 308 as the location of the requested data item. Thus, in step 332, the Put( ) command stores the requested data item in the remote cache 4. In this manner, the system has begun to balance the stored data items between all of the identified remote caches (e.g., the remote caches 1-4). As additional Get( ) commands are executed by the system, some requested data items stored in the remote caches 1-3 will be transferred to the remote cache 4 until the system 100 achieves some load balance between the remote caches 1-4. If the new remote cache (e.g., the remote cache 4) has a different storage capacity than the other remote caches (e.g., the remote caches 1-3), the mapping process described herein can be modified to balance the workload based on the relative storage capacities of the various remote caches. Such alteration to the mapping process can be accomplished by one skilled in the art without additional information. Thus, the system 100 advantageously allows new remote caches to be added to the system on the fly and achieves load balancing between the remote caches to accommodate the addition of the new remote cache(s).

Detailed Description Text (111):
The process of retrieving data from remote caches at different levels, such as the remote caches 1-9 illustrated in FIG. 10, is illustrated in the flowchart of FIG. 14. At a start 350, it is assumed that all remote caches have been initialized and are operational. For the sake of clarity, it is further assumed that there are no newly added remote caches that require load balancing, which is illustrated in the flowcharts of FIGS. 12 and 13. In step 352, the client 102 (see FIG. 5) generates a Get( ) command to retrieve a requested data item. As noted above, the Get( ) command includes a key to uniquely identify the requested data item.

Detailed Description Text (136):
Similarly, the lists B1-L1 of FIG. 18 each independently replicate the functionality of the list A1. That is, each list independently performs the function of the list A1 as described above with respect to FIGS. 16 and 17. In addition, each of the lists B1-L1 may also include multiple lists, such as lists

B2-L2 (not shown). It should be noted that the lists A1-L1 need not have the same number of higher priority utilization lists associated therewith. For example, the list A1 may have associated lists A2-A3, while the list B1 may only have an associated list B2. The system 100 can accommodate any variable number of lists associated with the lists A1-L1.

Previous Doc Next Doc Go to Doc#

L39: Entry 5 of 14 File: USPT Dec 26, 2000

DOCUMENT-IDENTIFIER: US 6167427 A
TITLE: Replication service system and method for directing the replication of information servers based on selected plurality of servers load

Application Filing Date (1):
19971128

Drawing Description Text (7):
FIG. 5 is a flowchart illustrating a load balancing system and method in accordance with one embodiment of the present invention.

Drawing Description Text (9):
FIG. 7 is a flowchart for a preferred illustrative load balancing arrangement.

Detailed Description Text (11):
It proves convenient to consider a mechanism for load balancing to be stabilizing if there exist two constants, demandDiff and loadDiff such that if the variation in time of the request rate for every object x stays within demandDiff, the system eventually reaches a state where no replicas are created or dropped, and the difference between the load of any two hosting servers does not exceed loadDiff.

Detailed Description Text (12):
Likewise, it proves advantageous to avoid load balancing of a type in which isolated regions are created with autonomous load distribution in each region. Thus, a load balancing mechanism desirably avoids a condition in which nodes from different regions have significantly different load, even if individual regions are not in the stable state. More formally, for some L and l<L, and with all nodes partitioned into three sets, A(L,l)={p.vertline.load(p).ltoreq.L}, B(L,l)= {p.vertline.l.ltoreq.load(p)<L}, and C(L,l)={p.vertline.load(p)<l}. A mechanism for load balancing is called contiguous if there exist some constants d and t such that for any L and l<L-d, if no node moves between sets A(L,l), B(L,l) and C(L,l) for time t, then there will be a distribution event either with a source in A and recipient outside A, or with a source outside C and recipient inside C.

Detailed Description Text (13):
The contiguity criterion may be illustrated by a system with four servers i, j, k, and l. A load balancing mechanism can be derived that balances the load between nodes in node pair i and j, and separately between nodes in node pair k and l, with no balancing performed between the node pairs. The demand for objects hosted by the first pair (i, j) will be assumed to be very high, but unstable (so that there are continuous distribution events occurring between i and j). Also, it will be assumed that there is unstable but overall low demand for objects on k and l. Thus the load on i and j greatly exceeds the load on k and l, while fluctuating within each pair. This example shows the importance of both the contiguity criterion as well as the stabilization criterion.

Detailed Description Text (17):
When a hosting server 140-j creates or drops a replica of an object, it records this fact at the name service 120. Thus, the name service keeps a mapping of the

symbolic name of an object to a set of physical names of currently available replicas. When resolving a symbolic name, the name service chooses one of the current set of physical names, applying a fair procedure and taking into account the geographical location of the requesting client.

Detailed Description Text (25):
A Preferred Load Balancing Protocol

Detailed Description Text (26):
Another system and method solution related to the above-described load balancing solutions that has been found to be particularly advantageous in a variety of network contexts is illustrated in FIG. 3. In this alternative and preferred embodiment, each replicator distributes load among subtrees rooted at its subordinates, and each subordinate accomplishes load distribution within its subtree. This solution will now be described in greater detail, initially with reference to FIG. 3.

Detailed Description Text (39):
If the same conditions appearing in FIG. 3 are used to direct the load-balancing embodiment reflected in Listing 1, then a successful redistribution of load will be accomplished. In particular, R3 will report load.sub.min,R3 =load.sub.min,Smin =5. R4 will report load.sub.max,R4 =6, and load.sub.max,R4 =8. Thus R2 will find load.sub.max,Smax -load.sub.min,Smin =6>4, and send Offload(p4) to p9. In addition, R2 will report the average load of all of its host descendants, load.sub.max,R2 =load.sub.min,R2 =5.33 (since it sent the Offload message). At the same time R1 will report load.sub.min,R1 =1 and load.sub.max,R1 =4, because it did not sent the Offload message. Therefore, R0 will discover load.sub.max,Smax -load.sub.min,Smin =4.33>4 and send Offload(p1) to p8.

Detailed Description Text (67):
First, an object may use a partial URL to organize a link to another object that resides on the same server. Partial URLs do not contain hosting server information--existing browsers by convention use the hosting server of the parent object to send the request for an object referenced by a partial URL. Suppose an object,foo.html, has a link to another object, bar.html. Moreover, it will be assumed that bar.html is not registered with the system, so that its physical URL is used. If bar.html resides on the same server, foo.html used a partial URL to refer to it. When the load balancing system of FIG. 1 moves foo.html to server2. abcd.com, the link to bar.html will become invalid; this server does not have such object.

Detailed Description Text (74):
According to the geographical extension to the basic protocols described above, a replicator R migrates or replicates reported objects among its subordinates to minimize the number of outside requests for the subordinates. The replicator then reports up to its parent the n objects reported to R which have the greatest number of requests from outside R's region. For the present geographical variant, it proves convenient to include in a determination of replica placement only immediate subordinate of a replicator, and not leaf servers. Thus replication placement decisions are reported by a replicator down through its subordinates instead of notifying leaf servers. Experience will dictate in particular cases how to resolve any possible conflict between improving load balancing and improving distance or cost factors associated with geography-based placement decisions.

Detailed Description Text (76):
The preceding discussion of methods and system for load balancing and improving geography or distance and message handling costs in information systems has, for simplicity of exposition, adopted an approach that starts with a small number of object replications and creates additional replicas as the demand for objects grows. Alternative embodiments, all within the scope of the present invention will

seek to have as many replicas of objects as possible, and to drop or delete replicas when the demand for space grows due to hosting for new objects. Other factors occasioning increased demand for space include the presence of objects that prove to be in very high demand. In any event, those skilled in the art will recognize that the presently disclosed methods and systems permit for the orderly reduction of replicas as required. Another variant for migrating high-demand objects is to associate such objects (including, as appropriate, lower-demand objects associated with one or more "hot" objects) for migration as a group.

Other Reference Publication (2):
Baumgariner et al., Global load balancing strategy for distributed computer system, IEEE, Jul. 1988.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L39: Entry 6 of 14 File: USPT Apr 18, 2000

DOCUMENT-IDENTIFIER: US 6052724 A
** See image for Certificate of Correction **
TITLE: Method and system for managing a directory service

Application Filing Date (1):
19970902

Brief Summary Text (17):
According to another embodiment of this invention, a second plurality of objects which are operative to maintain information about replicas associated with the plurality of partitions are defined in the MIB. The second plurality of objects can be maintained in a replica table object, each row in the replica table object containing information regarding a respective replica. Each replica row can contain a replica state object operative to maintain a state of the respective replica, and a replica type object operative to maintain a type of the respective replica.

Brief Summary Text (18):
The MIB according to one embodiment of this invention contains a partition table which maintains information about each partition on the managed server, and a replica table which maintains information about each replica associated with each partition in the partition table. Each managed server in the distributed directory service preferably has its own agent and associated MIB. By using standard network management protocols, such as the Simple Network Management Protocol (SNMP), the method and system according to this invention can be used with any network management station software suitable for use in an SNMP environment. Thus, the method and system according to this invention provide a uniform interface suitable for administering a complex distributed directory with standard network management software.

Detailed Description Text (19):
Server object 86 is a parent object to a plurality of child objects, such as partition object 88 and server info object 90, each of which are also group and thus parent objects to a plurality of child objects. Partition object 88 has a plurality of child objects, each of which relates to management of partitions on managed server 58. Referring now to FIG. 4, partition object 88 is a parent object to partition count object 91, partition table object 92, replica table object 93, referenced server table object 94, and partition operations table object 95. Partition count object 91 maintains the total number of partitions that reside on managed server 58. The formal ASN.1 syntax for partition count object 91 is as follows:

Detailed Description Text (29):
Referring to FIG. 6, replica table object 93 is a table which contains a plurality of rows, or sequences, of objects which relate to the replicas associated with the partitions identified in partition table object 92. This information includes not only replicas which exist on managed server 58, but also information regarding replicas on other network servers. Each row of replica table object 93 includes a partition replica ID object, a replica number object, a replica server ID object, a replica state object, a replica type object, a replica successful sync date time

object, a successful inbound sync count object, a successful outbound sync count object, a replica fail sync date time object, a replica sync fail condition object, a replica fail sync count object, a replica server state object, an inbound object count object, an outbound object count object, an inbound sync time object, and an outbound sync time object. The formal ASN.1 syntax, according to one embodiment of this invention, for replica table object 93 is provided below:

Detailed Description Text (30):
As with partition table object 92, the objects in the rows of replica table object 93 can be accessed from conventional management station software. For example, the replica state object can be queried to determine the current state of a particular replica, and the replica type object can be accessed to determine the type of the respective replica. The partition replica ID object identifies the partition associated with the respective replica, and the replica server status object can be queried to determine the status of the server on which the replica exists.

Detailed Description Text (37):
FIG. 11 shows a user interface window 150. Upon selection of item 152 of folder 130, window 151 displays information relating to the replicas associated with a particular partition. For example, box 153 contains a list of the replicas associated with the partition which was selected in FIG. 10. The replica information displayed in box 153 is obtained by traversing replica table object 93 and for each row, or sequence, in the replica table which relates to the selected partition, extracting information from the objects in that row. For example, the replica server name information shown in column 154 can be obtained by indexing the nwDSReferencedServerTable object with the Replica Server ID object obtained from the appropriate row of the Replica Table object, and extracting the server name from the nwDSServerName object. The replica state information shown in column 156 can be obtained from the replica state object in replica table 93. The replica type information shown in column 158 comes from the replica type object of replica table 93. The last sync time information shown in column 160 comes from the replica successful sync date time object in replica table 93.

Detailed Description Text (38):
As a replica is highlighted in box 153, such as replica 155, additional information about that selected replica can be provided, such as shown in box 161. Field 162 contains the last inbound synchronization time. This information can be obtained from the inbound sync time object in the row of replica table object 93 associated with the selected replica. Field 164 contains the outbound sync time. This information can be obtained from the outbound sync object of replica table object 93. Field 166 contains the date and time of the last synchronization failure. This information can be obtained from the replica fail sync date time object. Field 168 contains the current server state of the selected replica. This information can be obtained from the replica server state object of replica table object 93. Field 170 contains the replica type of the selected replica. This information can be obtained from the replica type object of replica table object 93. An administrator can change the type of replica by selecting one of the radio buttons presented in box 170 and activating button 149. Upon doing so, the appropriate object in replica table object 93 will be changed to reflect the new type of replica, and directory service 62 will be prompted to take the appropriate action.

Detailed Description Text (39):
FIG. 12 shows a user interface window 172. Window 172 contains three separate windows 175, 177 and 179. Window 177 contains a graphical display of icons which represent the various components in a network. Upon selecting a server icon, such as icon 176, window 175 displays one or more partition icons which represent each partition located on the selected server. This information can be obtained from partition table object 92. Upon selection of one of the displayed partitions, such as for example partition 174, window 179 displays a list of the containers on the selected partition. The container information shown in window 179 can be obtained

from the container usage table object 97 of server info object 90. Directory service 62 offers an API which translates a container ID into the partition name in which the container is located. Column 178 contains the name of each container in the selected partition. This information can be obtained from container usage table object 97. Column 180 contains the number of times each container has been read since directory service 62 was initiated. This information can be obtained from the container read operations object from the row associated with that container. Column 182 contains the number of times a write operation has been performed against the selected container. This information can be obtained from the container write operations object of the row associated with that container. Container usage information can be used by an administrator for load balancing and determining which containers are most heavily accessed.

Detailed Description Paragraph Table (1):

DS Partition Table nwDSPartitionTable OBJECT-TYPE SYNTAX SEQUENCE OF NwDSPartitionEntry ACCESS not-accessible STATUS mandatory DESCRIPTION "A list of all partitions that reside on this server." ::= { nwDSPartition 2 } nwDSPartitionEntry OBJECT-TYPE SYNTAX nwDSPartitionEntry ACCESS not-accessible STATUS mandatory DESCRIPTION "The description of a particular partition in the partition table." INDEX {nwDSPartitionID} ::= { nwDSPartitionTable 1 } nwDSPartitionEntry ::= SEQUENCE { nwDSPartitionID DsObjectID, nwDSPartitionName ObjectFullDistinquishedName, nwDSLastSuccessPartitionOperation INTEGER, nwDSLastSuccessPartOperStartDateTime DateAndTime, nwDSLastSuccessPartOperStopDateTime DateAndTime, nwDSCurrentOperation INTEGER, nwDSCurrentOperationStartDateTime DateAndTime, nwDSCollisionCount INTEGER, nwDSLastCollObjectName ObjectFullDistinquishedName, nwDSLastColReName ObjectFullDistinquishedName, nwDSLastCollisionDateTime DateAndTime, nwDSObjectCount INTEGER, nwDSReplicaPerPartitionCount INTEGER, nwDSLastEntryModTime DateAndTime, nwDSSapFederatedName DisplayString, nwDSStartSapNameMode INTEGER, nwDSPartReplicaNumber INTEGER nwDSPartitionID OBJECT-TYPE SYNTAX DsObjectID ACCESS read-only STATUS mandatory DESCRIPTION "The object ID which uniquely identifies a partition on this server and which maps as an index to the partition table." ::= { nwDSPartitionEntry 1 } nwDSPartitionName OBJECT-TYPE SYNTAX ObjectFullDistinquishedName ACCESS read-only STATUS mandatory DESCRIPTION "The Distinguished Name of this partition." ::= { nwDSPartitionEntry 2 } nwDSLastSuccessPartitionOperation OBJECT-TYPE SYNTAX INTEGER { idle(1), split(2), splitChild(3), joinUp(4), joinDown(5), moveSubTreeSource(6), moveSubTreeDestination (7), repairingTimeStamps(8), changingReplicaType(9) } ACCESS read-only STATUS mandatory DESCRIPTION "The last successful operation (such as delete, join, split, move, etc.) that was performed on this partition. Example: Deleted Partition1 The initial value is 0." ::= { nwDSPartitionEntry 3 } nwDSLastSuccessPartOperStartDateTime OBJECT-TYPE SYNTAX DateAndTime ACCESS read-only STATUS mandatory DESCRIPTION "The date and time when the last successful operation on this partition was initiated. The initial value is 0." ::= { nwDSPartitionEntry 4 } nwDSLastSuccessPartOperStopDateTime OBJECT-TYPE SYNTAX DateAndTime ACCESS read-only STATUS mandatory DESCRIPTION "The date and time when the last successful operation on this partition was completed. The initial value is 0." ::= { nwDSPartitionEntry 5 } nwDSCurrentOperation OBJECT-TYPE SYNTAX INTEGER { idle(1), split(2), splitChild(3), joinUp(4), joinDown(5), moveSubTreeSource(6), moveSubTreeDestination(7), repairingTimeStamps(8), changingReplicaType(9) } ACCESS read-only STATUS mandatory DESCRIPTION "The operation currently being performed on this partition (delete, join, split, move, etc.)." ::= { nwDSPartitionEntry 6 } nwDSCurrentOperationStartDateTime OBJECT-TYPE SYNTAX DateAndTime ACCESS read-only STATUS mandatory DESCRIPTION "The date and time when the current operation was initiated." ::= { nwDSPartitionEntry 7 } nwDSCollisionCount OBJECT-TYPE SYNTAX Counter ACCESS read-only STATUS mandatory DESCRIPTION "The total number of collisions that have occurred on this partition since NDSStats.NLM was loaded." ::= { nwDSPartitionEntry 8 } nwDSLastCollObjectName OBJECT-TYPE SYNTAX ObjectFullDistinquishedName ACCESS read-only STATUS mandatory DESCRIPTION "The report (by object name) of the last name collision." ::= { nwDSPartitionEntry 9 }

nwDSLastColReName OBJECT-TYPE SYNTAX ObjectFullDistinquishedName ACCESS read-only STATUS mandatory DESCRIPTION "The report (by renamed object) of the last name collision." ::= { nwDSPartitionEntry 10 } nwDSLastCollisionDateTime OBJECT-TYPE SYNTAX DateAndTime ACCESS read-only STATUS mandatory DESCRIPTION "The date and time when the last name collision occurred." ::= { nwDSPartitionEntry 11 } nwDSObjectCount OBJECT-TYPE SYNTAX INTEGER ACCESS read-only STATUS mandatory DESCRIPTION "The total number of objects in this partition." ::= { nwDSPartitionEntry 12 } nwDSReplicaPerPartitionCount OBJECT-TYPE SYNTAX INTEGER ACCESS read-only STATUS mandatory DESCRIPTION "The total number of replicas of this partition." ::= { nwDSPartitionEntry 13 } nwDSLastEntryModTime OBJECT-TYPE SYNTAX DateAndTime ACCESS read-only STATUS mandatory DESCRIPTION "The date and time when an entry in the partition table was last modified. Initial value is 0." ::= { nwDSPartitionEntry 14 } nwDSSapFederatedName OBJECT-TYPE SYNTAX DisplayString (SIZE(0..48)) ACCESS read-only STATUS mandatory DESCRIPTION "SAP and real federated names are the same." ::= { nwDSPartitionEntry 15 } nwDSStartSapNameMode OBJECT-TYPE SYNTAX INTEGER { on(1), off(2), rootmost(3) } ACCESS read-write STATUS mandatory DESCRIPTION "The setting for Service Advertising (SAP) on this federated partition (on, off, or rootmost). When the setting is rootmost, only the rootmost partition object advertises." ::= { nwDSPartitionEntry 16 } nwDSPartReplicaNumber OBJECT-TYPE SYNTAX INTEGER ACCESS read-only STATUS mandatory DESCRIPTION "The replica number, which also maps as an index to the partition's replica table." ::= { nwDSPartitionEntry 17 } .COPYRGT. 1997 Novell Inc.

Detailed Description Paragraph Table (2):

DS Replica Table nwDSReplicaTable OBJECT-TYPE SYNTAX SEQUENCE OF NwDSReplicaEntry ACCESS not-accessible STATUS mandatory DESCRIPTION "The list of all replicas of this partition that reside on other servers." ::= { nwDSPartition 3 } nwDSReplicaEntry OBJECT-TYPE SYNTAX NwDSReplicaEntry ACCESS not-accessible STATUS mandatory DESCRIPTION "The description of a particular replica (in the replica table) that resides on this partition." INDEX {nwDSPartitionRepID,nwDSReplicaNumber} ::= { nwDSReplicaTable 1 } NwDSReplicaEntry ::= SEQUENCE { nwDSPartitionRepID DsObjectID, nwDSReplicaNumber INTEGER, nwDSReplicaServerID INTEGER, nwDSReplicaState INTEGER, nwDSReplicaType INTEGER, nwDSRepSuccessSyncDateTime DateAndTime, nwDSSuccessInBoundSyncCount Counter, nwDSSuccessOutBoundSyncCount Counter, nwDSReplicaFailSyncDateTime DateAndTime, nwDSReplicaFailCondition INTEGER, nwDSReplicaFailSyncCount INTEGER, nwDSReplicaServerState INTEGER, nwDSInBoundObjectCount Counter, nwDSOutBoundObjectCount Counter, nwDSInBoundSyncTime Counter, nwDSOutBoundSyncTime Counter nwDSPartitionRepID OBJECT-TYPE SYNTAX DsObjectID ACCESS read-only STATUS mandatory DESCRIPTION "The object ID that uniquely identifies a particular replica on this partition. This ID also maps as an index to the partition's replica table." ::= { nwDSReplicaEntry 1 } nwDSReplicaNumber OBJECT-TYPE SYNTAX INTEGER ACCESS read-only STATUS mandatory DESCRIPTION "The number that uniquely identifies a particular replica on this partition. This number also maps as an index to this server's replica table." ::= { nwDSReplicaEntry 2 } nwDSReplicaServerID OBJECT-TYPE SYNTAX INTEGER ACCESS read-only STATUS mandatory DESCRIPTION "The object ID that identifies the server on which this replica resides. This number also maps as an index to the reference server table (nwDSRefServerID)." ::= { nwDSReplicaEntry 3 } nwDSReplicaState OBJECT-TYPE SYNTAX INTEGER { on(0), new(1), dying(2), locked(3), changereplicatype0(4), changereplicatype1(5), transition(6), split0(48), split1 (49), join0(64), join1(65), join2(66), move0(80) } ACCESS read-only STATUS mandatory DESCRIPTION "The state of this replica (busy, on, join, etc.)." ::= { nwDSReplicaEntry 4 } nwDSReplicaType OBJECT-TYPE SYNTAX INTEGER { master(0), readWrite(1), readOnly(2), subref(3) } ACCESS read-only STATUS mandatory DESCRIPTION "The type of this replica (master, read-write, read-only, or subordinate reference)." ::= { nwDSReplicaEntry 5 } nwDSRepSuccessSyncDateTime OBJECT-TYPE SYNTAX DateAndTime ACCESS read-only STATUS mandatory DESCRIPTION "The date and time when this replica was last successfully synchronized." ::= { nwDSReplicaEntry 6 } nwDSSuccessInBoundSyncCount OBJECT-TYPE SYNTAX Counter

ACCESS read-only STATUS mandatory DESCRIPTION "The total number of times that in-bound synchronization was successful since NDSStats.NLM was loaded." ::= { nwDSReplicaEntry 7 } nwDSSuccessOutBoundSyncCount OBJECT-TYPE SYNTAX Counter ACCESS read-only STATUS mandatory DESCRIPTION "The total number of times that out-bound synchronization was successful since NDSStats.NLM was loaded." ::= { nwDSReplicaEntry 8 } nwDSReplicaFailSyncDateTime OBJECT-TYPE SYNTAX DateAndTime ACCESS read-only STATUS mandatory DESCRIPTION "The date and time when the last failure of replica synchronization occurred." ::= { nwDSReplicaEntry 9 } nwDSReplicaSyncFailCondition OBJECT-TYPE SYNTAX INTEGER ACCESS read-only STATUS mandatory DESCRIPTION "The reason (error condition) that the last failure of replica synchronization occurred." ::= { nwDSReplicaEntry 10 } nwDSReplicaFailSyncCount OBJECT-TYPE SYNTAX Counter ACCESS read-only STATUS mandatory DESCRIPTION "The total number of times that synchronization has failed since NDSStats.NLM was loaded." ::= { nwDSReplicaEntry 11 } nwDSReplicaServerState OBJECT-TYPE SYNTAX INTEGER { unknown(0), down(1), up(2) } ACCESS read-only STATUS mandatory DESCRIPTION "The state of the server (up, down, unknown) on which this replica resides." ::= { nwDSReplicaEntry 12 } nwDSInBoundObjectCount OBJECT-TYPE SYNTAX Counter ACCESS read-only STATUS mandatory DESCRIPTION "The count of all in-bound objects that have been synchronized since NDSStats.NLM was loaded." ::= { nwDSReplicaEntry 13 } nwDSOutBoundObjectCount OBJECT-TYPE SYNTAX Counter ACCESS read-only STATUS mandatory DESCRIPTION "The count of all out-bound objects that have been synchronized since NDSStats.NLM was loaded." ::= { nwDSReplicaEntry 14 } nwDSInBoundSyncTime OBJECT-TYPE SYNTAX Counter ACCESS read-only STATUS mandatory DESCRIPTION "The total time (in seconds) required for in-bound objects to synchronize." ::= { nwDSReplicaEntry 15 } nwDSOutBoundSyncTime OBJECT-TYPE SYNTAX Counter ACCESS read-only STATUS mandatory DESCRIPTION "The total time (in seconds) for out-bound objects to synchronize." ::= { nwDSReplicaEntry 16 } .COPYRGT. 1997 Novell Inc.

CLAIMS:

7. A method according to claim 6, further comprising a replica table object, wherein the replica table object is operative to maintain a plurality of rows, each row being operative to maintain information for a respective replica and including one of the replica state objects and one of the replica type objects.

9. A method according to claim 2 further comprising a partition table object, wherein the partition table object is operative to maintain a plurality of rows for maintaining information about respective partitions on the managed server, each row containing one of the partition operation objects and one of the current operation objects, and a replica table object, wherein the replica table object is operative to maintain a plurality of rows, each row being operative to maintain information for a respective replica and including one of the replica state objects and one of the replica type objects.

19. A method according to claim 18, wherein the second plurality of objects comprise a plurality of replica state objects, each replica state object being operative to maintain a state of the respective replica, and a plurality of replica type objects, each replica type object being operative to maintain a type of the respective replica, the information base further comprising a replica table object, the replica table object begin operative to maintain a plurality of rows, each row being operative to maintain information for a respective replica and including one of the replica state objects and one of the replica type objects.

Previous Doc Next Doc Go to Doc#

DOCUMENT-IDENTIFIER: US 6052718 A
** See image for Certificate of Correction **
TITLE: Replica routing

Application Filing Date (1):
19970107

Brief Summary Text (4):
Certain known approaches for automatically directing client computers to servers include, for example, round robin DNS and loading balancing DNS, which direct users to one of a number of server replicas in an attempt to balance the load among many servers. In another approach called multiple hostnames, content is spread over multiple servers, each with a separate hostname. Web pages returned to users contain links that point at the replica that has been selected for the user based on load-balancing concerns and replica content considerations. In another approach called Internet load balancing, a hardware component automatically distributes user requests sent to a single IP address to one of a number of server replicas to implement load balancing. Another approach is resonate dispatch that combines load balancing with replica capability to automatically direct users to a replica that is operational, is not overloaded with requests, and contains the requested information.

Detailed Description Text (18):
At step 302 a server replica creates a network performance table using the method presented in FIG. 6. At step 305 the server replica creates a replica summary record that has one entry for each network it can reach. Each replica summary record entry contains: a network number, the network's net mask from the network performance table (see FIG. 6), an estimate of the performance to the network from the network performance table, and the current time as a timestamp. The entire replica summary record is marked as being created by a server replica.

Detailed Description Text (19):
Alternatively, if a particular computer is a replica router, then at step 306 the replica router scans its replica routing database and deletes any replica summary record entries that have a timestamp that is older than a configuration-set time limit. At step 307 a test is made to see if this replica router is configured as a root replica router. If so, then control is transferred to step 370 and the root replica router does not create an advertisement (because the root replica router has no parent to which it can send an advertisement). If this replica router is not a root replica router, then at step 310 the replica routing database is used to create a new replica summary record that has multiple entries, one for each network number advertised in a replica summary record in the replica routing database. Each entry in the newly created replica summary record includes: a network number, the net mask for that network number, the best performance metric value for that network number that is advertised in a replica summary record by any server replica or replica router, and the timestamp from this best performing entry in the routing database. The newly created replica summary record is marked as being created by a replica router.

Detailed Description Text (20):
At step 315 logic common to the replica routers and server replicas begins, and the new replica summary record can be modified according to operator-specific rules that are specific to the replica router or server replica. Arbitrary alterations to the new replica summary record can be specified, including: the removal of certain networks; the addition of network numbers with specified network masks, performance metric values, and timestamps that can include a "do not expire value"; manual override, by network number, of network masks and performance metric values or replica summary record entries; and removal of replica summary record entries that do not achieve a specified performance metric value. In this way the operator of a server can ensure that the server serves its intended audience, for example by adding intranet network numbers that cannot be seen from outside the intranet's firewall. Next, the replica router or server replica selects a set of parent replica routers. The addresses of the parent replica routers are initialized by looking up the replica routers bound to the service's DNS name (such as "www.pathfinder.com"). Alternatively, the set of parent replica routers can be manually configured for more involved hierarchies.

Detailed Description Text (23):
At step 325 a replica advertisement is constructed that includes the replica summary record, the IP address of the local computer, and the current time. A digital signature is added to the replica advertisement at 330 that is based upon a service-specific private key known to all replicas and replica routers of a service, and the completed replica advertisement is sent to the parent replica routers in HTTP POST message 335.

Detailed Description Text (25):
At step 342, if the IP addresses do not match, it means that the replica advertisement has traveled through a firewall (see FIG. 1). The multiple-entry replica summary record in the replica advertisement has a single entry added that includes: the source IP address in the header of message 335, a net mask of all bits "1," a default network metric value, and the current time. This additional entry is added to the summary because the added IP address will be identical to the IP addresses of requests made by clients from behind the same firewall, and thus will match the IP addresses of these client requests. This new replica summary record is marked as having been created by a replica router.

Detailed Description Text (28):
In an alternative embodiment, replica routers regularly "ping" the servers that are described by replica advertisements in their replica routing databases. Servers that do not respond have their advertisements removed unless their replica summary record contains an entry that has a net mask consisting entirely of "1"s, which indicates that the server is behind a firewall.

Detailed Description Text (30):
At step 535 the replica router matches all of the replica summary record entries in the replica routing database to the source IP address in message 515. Address matching of each replica summary record entry is performed based on the portion of each address that constitutes the network identifier according to the net mask contained in the entry. If no matches are found, control is transferred to step 562. If matches are found, at step 545 the N matching replica summary record entries that contain the best network performance metric values are selected and sorted by decreasing network performance metric value, and the IP addresses contained in the corresponding replica advertisement entries are made the candidate target IP addresses. Each entry in the candidate target IP address list includes a descriptor indicating whether it is a replica router or server replica (this information is determined from the entry's replica advertisement). The number N is a configuration parameter. Control then transfers to step 590.

Detailed Description Text (32):

At step 565 the IP address of each network router in the network route to the client is looked up in the replica summary records in the replica routing database, starting at the network router closest to the client. Matching is performed using the net mask in each replica summary record entry. If there are no matching entries in the replica routing database, then at step 575 a default set of pre-specified server replicas is made the set of candidate target IP addresses, and all of the default replicas are marked as server replicas. Control is then transferred to step 590.

Detailed Description Text (33):
If there are matching entries in the replica summary records in the routing database then at step 570 each matching replica summary record entry has its advertised network performance added to the network performance estimated from the client to the network router it matched. One way to estimate this performance is to take the round-trip performance observed from the replica router to the client and adjust for the round-trip performance from the replica router to the network router that matched. The N matching replica advertisement entries that contain the best aggregate network performance metric values are selected, and the IP addresses contained in these replica advertisement entries are made the candidate target IP addresses. Each entry in the candidate target IP address list includes a descriptor indicating if it is a replica router or server replica; this information is determined from the entry's replica advertisement. The N IP addresses are ordered by decreasing network metric merit. Control then transfers to step 590.

Detailed Description Text (39):
At step 630 a replica router uses the source IP address of message 620, and performs steps 535 to 570 or 575 from FIG. 4a to compute a set of candidate target IP addresses. In the place of step 562, the network performance table computed in step 610 and transmitted in message 620 is used to create an ordered list of network numbers reachable from the client in descending order of performance. These network numbers are looked up in the replica summary records in the replica routing database (step 565), and matching replica summary record entries result in an aggregate performance number being computed (step 570) from both the entry and the network performance provided by client message 620. The top N entries are used as candidate target IP addresses. Each entry in the candidate target IP address list includes a descriptor indicating whether it is a replica router or server replica (this information is determined from the entry's replica advertisement). If none of the networks in the network map in message 620 match replica summary records, then a default set of server replicas is used for the candidate target addresses (step 575).

Other Reference Publication (5):
Wallace, Bob; "Load balancing juggles calls at busy Web sites"; pp. 1-2, Mar. 17, 1997 or earlier; http://www.computerworld.com/search/AT-html/9611/961111SL46web.html.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L39: Entry 11 of 14 File: USPT Jul 28, 1998

DOCUMENT-IDENTIFIER: US 5787247 A
** See image for Certificate of Correction **
TITLE: Replica administration without data loss in a store and forward replication enterprise

Application Filing Date (1):
19960712

Brief Summary Text (9):
In deciding which servers should have copies of which data sets, an administrator typically considers such factors as load balancing, network topology, disk space, network bandwidth, replication latency, and frequency of access. An administrator will typically try to locate copies of the data sets in such a manner that the load on the network is balanced, and which will take advantage of plentiful network resources and minimize dependence on scarce network resources. Furthermore, factors such as replication latency, which represents the time needed for a change made to one copy of a data set to ripple through the network, should also be considered.

Brief Summary Text (24):
Data set properties: A set of information used to describe or track key information about a data set including a name and/or ID value and the replica list for the data set. The data set properties can also contain other information such as a change number, and a time stamp indicating the time last modified.

Brief Summary Text (27):
Replica list: A list of all replica nodes on the network containing a replica of a data set. Each entry in the replica list also contains the replica state for each replica node and a time stamp indicating when the entry was last modified.

Brief Summary Text (33):
In a store and forward replication process, each server keeps track of locally made changes to a particular copy of a data set (sometimes referred to as a "replica") which contains one or more data objects. Each server periodically broadcasts the new locally made changes (since the last replication broadcast) to all other servers with a copy of the data set. The group of other servers also having a copy of the data set is kept on a "replica list." The changes are preferably broadcast in the form of updated copies of the changed data objects. This allows each server to update its local copy of the data set as changes are received by replacing the older data objects with the newer data objects. In essence, a store and forward replication process utilizes existing networking hardware as a transport system to deliver one way, unacknowledged messages between servers, much like E-mail messages are delivered between users of computer systems.

Brief Summary Text (34):
In addition to replicating data set objects, store and forward replication also replicates "data set properties." Data set properties describe and define a data set. Data set properties can include such information as a data set name and/or other identifying information, access privilege information such as a list of users which can access or change the contents and/or properties of the data set, a

replica list which contains those replica nodes with a copy of the contents of the data set, and other information. Data set properties can be replicated independently from the data objects of the data set. By replicating data set properties to all or some of the replica nodes in the enterprise, those replica nodes will know what data sets are available throughout the enterprise and where copies of the data sets can be found.

Brief Summary Text (36):
Each data set has a replica list property, containing the list of replica nodes, their replica state and a time last modified stamp indicating the time that the replica state was last modified. In this way, when the replication process replicates data set properties throughout the network, then servers will know the replica state of the other servers in the network for each data set.

Detailed Description Text (18):
The replica list is the list of replica nodes having a copy of a populated version of the data set. The replica list acts as a distribution list for replication packets containing changes to the contents of a data set. The replica list may also contain other information, such as a replica state indicating the level of participation of each replica node on the list in the replication of the data set and a time last modified stamp indicating the time that the replica state was last modified.

Detailed Description Text (33):
A set of properties may be associated with each data set. These properties, often referred to as "data set properties," include such information as a name and/or other identifying information, access control information such as a list of users which can access the contents of a data set or the properties of the data set, and a list of replica nodes (a replica list) which have the contents of a data set. In addition, data set properties may include other information as described in greater detail below.

Detailed Description Text (34):
Because data set properties are associated with each data set, the data set properties provide a convenient location to store information which is needed by the present invention. As described in greater detail hereafter, the present invention utilizes a "replica state" to indicate the level of participation of each replica node in the data replication of each data set. For example, the replica state could indicate that a replica node was actively participating in the replication for a particular data set. In addition, the replica state could indicate that the replica node had deleted its local copy of the data set. Finally, the replica state could indicate that the replica node was in the process of deleting its local copy of the data set. In fact, a replica state can be established for any level of participation in the replication of a particular data set. The replica states defined in this application are designed to fulfill the goals of replica administration without data loss. The replica states allow such administration functions as adding a replica, deleting a replica, or moving a replica, while preventing inadvertent data loss and fulfilling the goals of the present invention. Embodiments within the scope of the present invention can comprise means for tracking the replica state of a replica node. By way of example, and not limitation, such means can comprise replica list 34.

Detailed Description Text (35):
Since the store and forward replication process maintains a replica list that contains all the replica nodes which are participating in the replication of a particular data set, the replica list provides a convenient location to store replica state information. One such replica list is illustrated in FIG. 2 as replica list 34. In one preferred embodiment, each entry in the replica list comprises: ##STR2##

Detailed Description Text (36):
The replica node ID is an identifier which uniquely identifies a particular replica node in the enterprise. Any type or form of ID value may be utilized for the replica node ID as long as the identifier is capable of uniquely identifying a particular replica node in the enterprise. One suitable method for generating a unique replica node ID is summarized later in this application and is more particularly disclosed in the copending Store and Forward Application, previously incorporated by reference. The replica state is the current replica state of the replica node identified by the replica node ID. The time last modified is a time stamp value which indicates the time that the replica state was last modified. The time last modified stamp is utilized in replica list maintenance, discussed below.

Detailed Description Text (37):
Because the present invention utilizes the replica list to store information about the replica state of a particular replica node with regards to a particular data set, and because such replica lists are replicated around the enterprise by replication processing block 26, care must be taken to properly maintain the replica lists, such as replica list 34. In FIG. 2, such a maintenance function is performed by replica list maintenance block 36. Replica list maintenance block 36 is responsible for ensuring that replica list 34 is properly updated as new information is received via replication processing block 26. Such updating includes the resolution of any conflicts which may arise during the replication of replica lists.

Detailed Description Text (38):
Because data set properties, including replica lists, are replicated across the enterprise, and because changes may occur to these data set properties at any replica node, a situation may arise where one copy of the data set properties on one replica node is changed so as to be in conflict with another copy of the data set properties on another replica node. For example, suppose the data set properties included a data set name which could be changed by a user. Now suppose user 1 connected to one replica node changed the name of a data set from A to B. Suppose that, simultaneously, user 2 connected to a different replica node changed the data set name from A to C. In this situation, a conflict between the names would arise and would have to be resolved when copies of the data set properties were exchanged during replication. Similar conflicts can arise with regard to other data set properties in the replica list. Any conflicts which arise must be resolved in a manner that does not create problems for the present invention.

Detailed Description Text (39):
Replica list maintenance block 36 can be part of the present invention, or may be part of another component of the replication process. In essence, all that is required by the present invention is that replica list 34 is updated so that the most recent entries reside in replica list 34 and so that no entries are lost during any conflict resolution process which is utilized. These two requirements may be met by utilizing the time last modified time stamp of the replica list. As previously described, the time last modified time stamp indicates the time which the replica state was list modified. When new information is received via replication processing block 26, replica list maintenance block 36 should compare the received replica list with replica list 34. As corresponding replica list entries are compared, the replica list entry with the later time stamp may be taken as the most current entry. Furthermore, entries having no corresponding entry in the other replica list can be merged into the final list. For example, consider replica list 1 which has entries as follows:

Detailed Description Text (40):
and replica list 2 which has entries:

Detailed Description Text (42):
When the lists are merged, the A entry is identical in both tables and so it is

kept without change. The D entry of list 2 has a time stamp of T.sub.4 which is later than the D entry of list 1 which has a T.sub.2 time stamp. Thus, the D entry of list 2 is taken and the D entry of list 1 is eliminated. The G entry exists only in list 1 and has no corresponding entry in list 2 so the G entry of list 1 is included in the final updated list. A much more detailed algorithm for detecting and resolving conflicts between different copies of the properties of a data set is disclosed in the copending Conflict Resolution Application, previously incorporated by reference. The conflict resolution process as disclosed in the Conflict Resolution Application is suitable for use in replica list maintenance block 36.

Detailed Description Text (43):
When a conflict is recognized and resolved by replica list maintenance block 36, it may be desirable to inform one or more users that a conflict has been recognized and resolved. Embodiments within the scope of the present invention may comprise means for notifying one or more users of a conflict. In FIG. 2 such means is illustrated, for example, tby conflict notification packet 37. The information in conflict notification 37 can be simple, such as a notification that a conflict was noted and resolved, or can be more detailed, such as a detailed explanation of what the conflict included and how it was resolved. Also, notification may be sent to any number of individuals. For example, it may make sense in some implementations to send notification to an "owner" of the data set and to the individuals which made the changes that created the conflict. Other combinations are possible.

Detailed Description Text (46):
As defined in greater detail below, state transition is initiated when a certain event occurs or when one or more conditions are satisfied. In FIG. 2, as such events or conditions occur, the replica states stored in replica list 34 may be updated or modified by local processing block 38 or replica state monitoring/update block 40. As the replica states are updated and stored in replica list 34, replication processing block 26 will distribute the changes to other replica nodes in the enterprise. Thus, other replica nodes in the enterprise will have an updated replica list illustrating the current replica state of the replica nodes on the replica list.

Detailed Description Text (48):
As disclosed in greater detail hereafter, before a copy of the data set is removed from a replica node, in order to guard against inadvertent data loss a predefined series of steps are taken. In one embodiment, the series of steps includes forcing replication of changes which have been made to local replica 32 and/or the data set properties associated with replica 32 (including replica list 34). Embodiments within the scope of this invention can therefore comprise means for initiating replication of changes which have been made but which have not yet been previously replicated. In FIG. 2, such means can comprise, for example, control 42. As illustrated in FIG. 2 control 42 provides a mechanism for replica state processing block 44 to direct replication processing block 26 to initiate replication of the changes which have been made but which have not yet been replicated.

Detailed Description Text (60):
Returning for a moment to FIG. 2, as previously described replica list 34 is preferably part of the data set properties which are replicated across the enterprise by replication processing block 26. In one preferred embodiment, replication processing block 26 represents a store and forward replication process. As previously described, the store and forward replication process will replicate copies of data sets to replica nodes on the replica list for that particular data set. Additionally, in one preferred embodiment, data set properties are replicated to every replica node in the enterprise. This means that all replica nodes in the enterprise have available to them the replica list for each data set which is being replicated across the enterprise. Each replica node in the enterprise therefore knows which data sets are being replicated across the enterprise, which replica nodes have copies of the data sets, and the replica state for each replica node on

each replica list.

Detailed Description Text (61):
If each replica node on the enterprise has available the replica state for the replica nodes involved in replicating each data set across the enterprise, then the replica state can be used to initiate a change from one replica state to another replica state. In one preferred embodiment, a system administrator does not change the replica state directly but only issues add or remove commands to change the replica state information in replica list 34 for a particular replica node. In FIG. 2, such a change can occur via local processing block 38. If the administrator generates a state change event (such as add replica event 54 or remove replica event 56 of FIG. 3), then the state information in replica list 34 can be changed by local processing block 38. If the replica state information in replica list 34 pertains to the local replica node, then replica state monitoring/update block 40 will recognize that a state change has occurred and will notify replica state processing block 44 to take appropriate action. If, however, the change made to the replica state pertains to another replica node in the enterprise, then when the data set properties (including replica list 34) are replicated via replication processing block 26 and received by the appropriate replica node, its replica state monitoring/update block will recognize the state change and also take appropriate action.

Detailed Description Text (62):
Replicating data properties in this manner creates two fundamental advantages. First it allows each replica node to receive any changes made to their replica state regardless of where the change originates. This allows an administrator to perform replica administration for any replica node while connected to any other replica node. The second advantage is that by replicating data set properties, including replica state information, each replica node knows how to treat other replica nodes. For example, if a user is connected to a replica node which does not have a local copy of a particular data set that the user wishes to access, then the replica node can check the replica list for that data set and discover which replica nodes have an active copy of the data set. Access to the data set is thus gained indirectly.

Detailed Description Text (71):
6. Remove the replica from the replica list.

Detailed Description Text (74):
When a replica node is in active state 52 and receives a remove replica event, remove replica event 56 indicates that the replica node transitions from active state 52 to delete pending state 62. As previously described, this may occur when replica state monitoring/update block 40 of FIG. 2 recognizes a state change for the local replica node in replica list 34 and passes that information to replica state processing block 44 of FIG. 2. When the replica node transitions to delete pending state 62, the replica node will perform the steps 2-4 listed above: terminate user access to the replica; flush unreplicated changes; and verify that the local data set contents reside at another replica node. In addition, if the transition from the active state to the delete pending state was made locally, the replication engine must also broadcast the locally made change to the data set's replica list. The initiation of the replica removal process occurs when the transition to delete pending state 62 is initiated.

Detailed Description Text (77):
As previously described, users may access a data set either locally or indirectly through another replica node. Because it is likely that copies of a particular data set only reside at certain locations in the network, then all access to that data set must be at one of the locations where a copy of the data set resides. If a user is connected to a replica node which does not have a copy of the data set, then the replica node will access a copy of a data set on another replica node in the

enterprise. This creates a situation where access to a particular Copy of the replica node may either be locally, for users connected to the local replica node, or remotely, for users connected to other replica nodes not having a copy of the data set. When terminating user access to a copy of a data set, both types of access must be terminated. This may be accomplished in several ways. One way is for the local replica node to refuse connections to the local copy of the data set. Access requests from other replica nodes can then be returned as unfillable because access is prevented. Local access can be prevented in much the same way. Indirect access to the local replica may also be prevented by screening access requests at the remote replica node. Since the replica state information is preferably available at all replica nodes, each replica node will know when the replica at another replica node enters the delete pending state. Access requests to that replica node can then be halted. An error message can then be sent to users which are accessing that copy of the data set. The users can then close the data set and reopen it to establish a connection to another valid copy of the data set. In a preferred embodiment, access requests to a data set or the contents of a data set, always check the replica list property to see if the folder is accessible.

Detailed Description Text (102):
There are situations and times where it is desirable to allow an administrator to bypass the safeguards put in place by the present invention, as for example if a replica node becomes stuck in delete pending state 62 of FIG. 3. Embodiments within the scope of this invention can, therefore, comprise means for bypassing the verification process so that the local copy of the data set can be deleted without verifying that the changes reside on at least one other replica node. In FIG. 3, such means is indicated by remove replica event 56. As previously described, an administrator can issue a command to generate a remove replica event. If such an event is received while the replica node is in delete pending state 62 of FIG. 3, then remove replica event 56 indicates that the replica node would transition from delete pending state 62 to delete now state 64. Although not shown explicitly in FIG. 4, if a remove replica event is received, then processing is suspended and the delete now state is entered. A remove replica event would cause the replica state in replica list 34 of FIG. 2 to change to "delete now" (in accordance with FIG. 3). Replica state monitoring/update block 40 could then inform replica state processing block 44 of the state change. Replica state processing block 44 can then perform the relevant processing.

Detailed Description Text (103):
As indicated in FIG. 3, delete pending state 62 may also be exited and delete now state 64 entered when the changes are verified. This is indicated in FIG. 3 by changes verified event 58. Returning now to FIG. 4, if the changes in the local copy of the data set reside elsewhere in the enterprise, decision block 74 indicates that the next step would be to initiate a transition to the delete now state. This is illustrated in FIG. 4 by step 78. When this step is reached, replica state processing block 44 of FIG. 2 could direct replica state monitoring/update block 40 to change the replica state in replica list 34 to "delete now." Replica state processing block 44 could then transition to the delete now state.

Detailed Description Text (104):
Embodiments within the scope of this invention can, therefore, comprise means for initiating transition from one intermediate state to another intermediate state. Such means may comprise, for example, changes verified event 58 or remove replica event 56 of FIG. 3. Such means may also comprise, for example, the structures of FIG. 2 which initiate a state transition by modifying the replica state in replica list 34 such as local processing block 38 and replica state monitoring/update block 40.

Detailed Description Text (109):
As with previous state transitions, replica state processing block 44 of FIG. 2 can inform replica state monitoring/update block 40 to modify the replica state of

replica list 34 from delete now to deleted. It is apparent from FIG. 5 that the delete now state is relatively short lived and transitory. This means that once the replica state of replica list 34 in FIG. 2 is modified, the likelihood of that state being replicated to other replica nodes via replication processing block 26 is small. However, if such a state is replicated to other replica nodes in the enterprise, then no unforeseen effects need be created. If other replica nodes in the enterprise treat the delete now state just as they would treat either the delete pending state or the deleted state, then operation of the replication environment should be predictable and consistent with whichever state is selected.

Detailed Description Text (110):
When deleted state 66 of FIG. 3 is entered, very little processing need be performed. The processing of deleted state 66 is illustrated in FIG. 6. As illustrated therein, only a single processing step is performed. As illustrated in step 86, when deleted state 66 of FIG. 3 is entered, the entry in the replica list is set to the deleted state. As previously described in conjunction with other replica state changes, replica state processing block 44 of FIG. 2 can direct replica state monitoring/update block 40 to modify the replica state in replica list 34 to deleted. This deleted state will then be replicated via replication processing block 26 to other replica nodes in the enterprise.

Detailed Description Text (116):
As previously discussed, add replica events are typically initiated by an administrator. An administrator directs that a copy of a data set be located on a particular replica node. Local processing block 38 of FIG. 2 can then respond to the add replica event by modifying the replica state in replica list 34 to active. If the local replica node is being moved to the active state, then replica state monitoring/update block 40 will note the state change in replica list 34 and inform replica state processing block 44 which can then take appropriate action. On the other hand, if the replica state change pertains to another replica node, when replica list 34 is replicated via replication processing block 26, the other replica node will receive the state change and take similar actions.

CLAIMS:

3. A method of removing a copy of a data object from a replica node as recited in claim 1 further comprising the step of removing the local replica node from a replica list comprising all replica nodes that have a copy of the data object.

10. A method of removing a copy of a data object from a replica node as recited in claim 9 further comprising the step of removing the local replica node from a replica list comprising all replica nodes that have a copy of the data object.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L43: Entry 4 of 28 File: USPT Oct 5, 1999

DOCUMENT-IDENTIFIER: US 5963944 A
** See image for Certificate of Correction **
TITLE: System and method for distributing and indexing computerized documents using independent agents

Application Filing Date (1):
19961230

Detailed Description Text (46):
Since there is no central coordinating computer to direct the replicating agents' movement among network nodes, sampling methods may be employed to track the number of replicas of each index file. For example, if there are 50 nodes in the network and each index file type should have five replicas, the replicating agent can be programmed to generate new replica of an index file if it fails to detect at least one index file of a given type after visiting 10 nodes. This can be done using counters for each index file type to track the number of nodes visited and the number of replicas detected in these visits

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit

Generate Collection

L43: Entry 21 of 28 File: DWPI May 9, 2000

DERWENT-ACC-NO: 2000-564279
DERWENT-WEEK: 200052

6,061,740

TITLE: Computer system for managing heterogeneous network through distributed directory has replication agent receiving messages from event monitor to modify set of objects of application server

PF Application Date (1):
19961209

PF Application Date (2):
19970715

Standard Title Terms (1):
COMPUTER SYSTEM MANAGE HETEROGENEOUS NETWORK THROUGH DISTRIBUTE DIRECTORY REPLICA AGENT RECEIVE MESSAGE EVENT MONITOR MODIFIED SET OBJECT APPLY SERVE

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit

Generate Collection

L43: Entry 19 of 28 File: DWPI Oct 25, 2001

2001/0034728

DERWENT-ACC-NO: 2002-033755
DERWENT-WEEK: 200272

TITLE: Electronic data files backing up system in e.g. computer system, has mirroring agent application to monitor electronic data in storage device of one node, periodically and transmit data to storage medium of other node

Basic Abstract Text (1):
NOVELTY - Several nodes referring to any electronic device are coupled through a communication network. Each node includes a mirroring agent application executed by a processor in that node. The agent application monitors original electronic data stored in data storage device of that node periodically and transmits the data to data storage medium in other nodes.

PF Application Date (1):
19990414

Previous Doc Next Doc Go to Doc#